STAGS User Manual

Charles C. Rankin
Rhombus Consultants Group, Inc., Palo Alto, California

William A. Loden
Resolutions, Cupertino, California

Francis A. Brogan

Harold D. Cabiness

April 2009
Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the lead center for NASA’s scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA’s institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA’s mission.

Specialized services that complement the STI Program Office’s diverse offerings include creating custom thesauri, building customized databases, organizing and publishing research results … even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at http://www.sti.nasa.gov
- E-mail your question via the Internet to help@sti.nasa.gov
- Fax your question to the NASA STI Help Desk at (301) 621-0134
- Phone the NASA STI Help Desk at (301) 621-0390
- Write to:
  NASA STI Help Desk
  NASA Center for AeroSpace Information
  7121 Standard Drive
  Hanover, MD 21076-1320
Preface

This User Manual is the largest of three reference documents for the current version (version 5.0) of the STAGS (STructural Analysis of General Shells) structural analysis program. The primary purposes of this document are (a) to describe the basic capabilities and features of the STAGS program and the most important post-analysis processors that can be used with it, and (b) to describe the input requirements for and the output generated by each of these programs. The other two reference documents for the current version of STAGS are the STAGS Elements Manual and the STAGS Test Cases Manual.

This document is an extensively updated version of large portions of the STAGS User Manual that was released when version 4.0 of the STAGS program was “frozen” in May of 2001. Many of the updates in this document describe new capabilities that have been introduced into STAGS since that date—new analysis features, 18-node sandwich elements, higher-level features of user-developed elements, and superelements, for example. Other updates in this document correct some of the errors and omissions that found (or failed to find) their way into earlier descriptions of older STAGS capabilities.

The first version of this User Manual document was released when version 2.0 of the STAGS program was released in June of 1994. That document was revised repeatedly until mid-1997, when it was “frozen” for version 2.4 of STAGS program. The STAGS 2.4 User Manual was revised repeatedly from mid-1997 through May of 2001, as STAGS 2.4 evolved into STAGS 3.0 and then into STAGS 4.0. This document is our “best shot” at describing STAGS 5.0.

Early versions of this User Manual were written for off-line use as printed documents. The current version of this document still has a very strong “printed reference” orientation; but the publication and distribution of this document in Adobe’s PDF format give it additional, platform-independent, on-line utilities that the earlier versions lacked. PDF versions of this document, the STAGS Elements Manual document and the STAGS Test Cases Manual document can be read and printed with Adobe’s Acrobat Reader application—which is widely-available, easy-to-use and free.
It is appropriate to digress briefly here to tell the reader that the STAGS program has a long and illustrious history. The STAGS program started out and in many important respects is still on the cutting edge of structural analysis technology. The earliest versions of what became the STAGS program (in the mid- to late-1960’s) were energy-based, finite difference codes that were designed to perform linear and nonlinear analyses of single-unit and multi-celled thin shell structures—principally (but not exclusively) for structures of interest to the aerospace and military communities. The earliest versions of STAGS were research-oriented, special-purpose programs that were developed on and designed to exploit the high-speed, large-problem computational capabilities of the CDC-6000-series mainframe computer systems that were the cream of the crop in the computer world during that time frame. During the early 1970’s, STAGS was “generalized” (to treat shell and other configurations that cannot easily be generated analytically) and converted into a research-oriented, general-purpose finite element program. During that decade and well into the second lustrum of the next, development of STAGS was centered on and designed to exploit the higher-speed, larger-problem capabilities of the CDC 7600 and the CRAY XMP, YMP and TS mainframe systems that were the performance champions of that era.

The 1.0 version of STAGS was an (early 1970’s) evolutionary extension of the special-purpose, research-oriented STAGS C-1 program). STAGS 2.0 was an (early 1980’s) evolutionary extension of STAGS 1.0. STAGS 2.0 evolved into STAGS 2.1, 2.2, 2.3, 2.4, 3.0, 3.5 and 4.0 over the next decade or so (from the mid-1980’s to the midpoint of 2002). The most recent (current) version of the program is STAGS 5.0.

The computer world has changed significantly over the last two decades, and STAGS has changed with it. “Omnia mutantur, nos et mutamur in illis.” Large, powerful mainframe systems were challenged by and eventually yielded to smaller (but still powerful) mainframe alternatives (the Convex and Stardent systems, in particular). These in turn were challenged by and yielded to still-smaller (and increasingly powerful) desktop workstation systems (by DEC, HP, IBM, SGI and SUN, principally). Versions of STAGS were developed for, and have been and continue to be effectively used on all of those machines. The shift from huge to smaller and smaller computer systems continues—with today’s desktop workstations being strongly challenged by today’s increasingly more powerful (and affordable) “personal” computer systems. Over the last several years, STAGS 4.0 and 5.0 have been transported to and are now being utilized effectively on many Linux, Windows and Macintosh-based personal computer systems.

Returning from that digression to the main theme of this Preface, we note again that the earliest versions of STAGS were research-oriented, special-purpose programs that evolved from there into the user-oriented, general-purpose STAGS system that we have today. The primary efforts in developing STAGS during the early and middle years (from the late 1960’s to the early 1990’s)
were very strongly focused on implementation of stat-of-the-art computer and innovative analysis technologies, with little energy (or funding) remaining for generating usable, up-to-date documentation. During those years, the STAGS User Manual was a thin, terse document that was updated from time to time (energy and funding permitting) to keep pace (as well as it could) with the program as it evolved. This was “tolerated” during those years because STAGS was viewed then as a cutting-edge tool that could be used most effectively by a relatively small number of computationally sophisticated researchers and analysts. This “once but no longer tolerable” documentation situation changed significantly when STAGS 1.0 evolved into STAGS 2.0, and the much-improved User Manual for STAGS 2.0 was developed to fill this need. Significant efforts have been made to update and improve that breakthrough User Manual to stay as up-to-date and as close as possible with STAGS and its companion programs during their evolutions from there to and including today’s STAGS 5.0 system.

It is also appropriate to note here that compatibility with STAGS C-1 was a major design consideration in the development of STAGS 1.0 and STAGS 2.0—and that maximization of compatibility with the most recent versions of STAGS has been a continuing consideration with each successive version of the program. With each new program capability and feature, however, changes to model-definition and solution-control input data have been inevitable. Strong efforts have been made to minimize those changes and keep them evolutional rather than revolutionary.

Input-data requirements for model definition and solution control are described in detail in Chapters 5 through 11 of this document and are summarized in one-line-per-record memory joggers in Appendix B. Input requirements for utilizing user-written subroutines in model-definition and analysis operations are described in Chapter 12. The basic approach that STAGS supports for implementing and utilizing user-defined elements is described in Chapter 13.

Users of earlier versions of the STAGS program and of the post-analysis processors that can be used in conjunction with STAGS should check the appropriate documentation very carefully to determine changes that are necessary before attempting to use any existing input files or attempting to employ user-written subroutines with version 5.0 of STAGS. We strongly suggest that, to determine where input changes are necessary and to discover new program capabilities and input features that make STAGS 5.0 easier to use and faster, users of earlier versions of the STAGS program should compare this document with the earlier documentation before using STAGS 5.0.

This User Manual is a reference document with a lot of gory detail in it. It is most appropriately used off-line, in its printed form; but it is also accessible and can be used on-line via Acrobat Reader. The STAGS Elements Manual contains more detailed descriptions of many (but not all) of the standard “spring” elements, “beam” elements, triangular and quadrilateral shell elements,
solid elements, shell/solid sandwich elements and “contact” elements that are implemented in STAGS 5.0 and its post-analysis processors. The STAGS Elements Manual also contains detailed descriptions of the input and interface requirements for implementing and utilizing user-developed elements in STAGS 5.0. The STAGS Test Cases Manual contains capsule descriptions of STAGS’ basic and special-purpose capabilities and in-depth descriptions of a number of illustrative and instructive test cases that are of interest to new and experienced STAGS users. The STAGS Test Cases Manual is an interesting and a very promising “work in progress.”

We also recommend the STAR Reference Manual as another valuable off-line companion to this User Manual and its other input/output-oriented companions. The STAR Reference Manual documents the STAGS Access Routines (STAR)—a library of FORTRAN routines that gives the user access to the STAGS Virtual Database (VDB). Major features of STAGS 5.0, STAR and VDB combine to give more sophisticated STAGS users high-level read/write access to model and solution data—facilitating the development of software for interfacing STAGS with other computational tools (such as graphical pre-/post-processors).

STAGS 5.0 is the most recent result of an ambitious project to enhance and rewrite large portions of the STAGS software and documentation. Development of STAGS and its companion programs continued under the leadership of Dr. Charles C. Rankin at the Lockheed–Martin Advanced Technology Center in Palo Alto, California—until his retirement from that company in April of 2003. Development of STAGS and its companion programs continues under his leadership now at the Rhombus Consultants Group in Mountain View, California. Additional code and documentation releases are anticipated in the future as more improvements are made.

Using this Document and its Companion Documents

Basic conventions used throughout this document for input-record descriptions are described in Chapter 5. Methods for navigating within the printed version of this document are described there and in the Table of Contents. The same record-description conventions and navigation methods used in this document are also used in the companion STAGS Test Cases Manual.

The most important navigation decisions that the analyst who is trying to use STAGS must make are usually related to the question “what is the next input requirement that must be satisfied?” These questions are answered by the “where to go from here” information that is at the end of each input-record description. For the example that is described in Chapter 5, the “where to go from here” information looks like this:
NESP (I-1) number of points on the (σ, ε) curve

if (NESP > 0) then go to I-3
else follow instructions at end of I-3a

The symbol, here, tells the user that important navigation information follows.

The key parts of this navigation information are in FORTRAN-like statements that tell the user which model-definition or solution-control record to consider next. More often than not, the next record depends on the value of one or more parameters on the current record and/or on previous records. Each parameter that was specified on a previous record is listed before the if-then-else constructs—the record identified by its alphanumeric index (I-1, in this example) in parentheses. The next record that the analyst must consider is identified by the pseudo-code that follows. In this example, the analyst is directed to the I-3 record if he/she gave a positive value to NESP on the most recent I-1 record, or to the navigation instructions at the end of the I-3a record if NESP was not positive.

The analyst who is using Acrobat Reader to view this document and/or its companion documents can transfer immediately to any page in the document being viewed with the “bookmark” facility that the Acrobat Reader application provides—and can also move around freely within that document using any of Reader’s other navigation and text-searching facilities.
Table of Contents

Preface ................................................. i

1 Introduction ........................................ 1-1
   1.1 About this Manual ................................ 1-1
   1.2 About STAGS .................................... 1-1

2 Installation and Execution ......................... 2-1
   2.1 STAGS Directory Structure (all platforms) .... 2-3
   2.2 UNIX-Based Systems .............................. 2-6
      Basic UNIX System Requirements .................. 2-6
      Installation of STAGS on a UNIX System ......... 2-6
      Execution of STAGS on a UNIX System .......... 2-10
   2.3 Linux-Based Systems ............................. 2-17
      Basic Linux System Requirements ............... 2-17
      Installation of STAGS on a Linux System ....... 2-17
      Execution of STAGS on a Linux System .......... 2-20
   2.4 Macintosh-Based Systems ....................... 2-26
      Basic Macintosh System Requirements .......... 2-26
      Installation of STAGS on a Macintosh system ... 2-26
      Execution of STAGS on a Macintosh System .... 2-29
   2.5 Windows-Based Systems ......................... 2-35
      Basic Windows System Requirements ............. 2-35
      Installation of STAGS on a Windows System ..... 2-35
      Execution of STAGS on a Windows System ....... 2-35
   2.6 Installation Verification ......................... 2-36
Table of Contents

3 Getting Started
   3.1 Infinite Cylinder Under Hydrostatic Pressure ........................................... 3-2
   3.2 Linear Bifurcation Buckling ................................................................. 3-4
   3.3 Large-Deflection Analysis of an Imperfect Cylinder .................................. 3-6
   3.4 Restarting a Nonlinear Analysis ............................................................. 3-8
   3.5 File Systems .............................................................................................. 3-8
   3.6 Input Files ................................................................................................ 3-10
       Linear buckling analysis ............................................................................ 3-11
       Start nonlinear analysis .......................................................................... 3-12
       Restart nonlinear analysis ...................................................................... 3-12
   3.7 Test problems ........................................................................................... 3-13

4 Fundamentals
   4.1 Coordinate Systems .................................................................................. 4-1
   4.2 Shell Unit ................................................................................................... 4-8
   4.3 The Element Unit ....................................................................................... 4-10
   4.4 Assembled Structures .............................................................................. 4-10
   4.5 Boundary Conditions ............................................................................... 4-10
   4.6 Loads .......................................................................................................... 4-11
   4.7 Summary of Modeling Techniques ............................................................ 4-13

5 Model Input
   5.1 Conventions Used in Input-Record Descriptions ........................................... 5-2
       Record ID and title .................................................................................... 5-2
       Discussion ................................................................................................ 5-2
       Data summary ........................................................................................... 5-2
       Data description ......................................................................................... 5-2
       Where to go from here .............................................................................. 5-3
   5.2 Example Input-Record Description .............................................................. 5-3
       Record ID and title: ................................................................................... 5-3
       Discussion: ................................................................................................ 5-3
       Data summary: ........................................................................................... 5-4
       Data description: ....................................................................................... 5-4
       Where to go from here: ........................................................................... 5-4
   5.3 Input Format Conventions ........................................................................... 5-4
   5.4 Summary and Control Parameters ............................................................... 5-6
   5.5 Discretization and Connectivity Summary .................................................. 5-22
   5.6 Data Tables ................................................................................................. 5-57
   5.7 References for Chapter 5 ............................................................................. 5-146
# Table of Contents

## 6 Model Input—Shell Units
- 6.1 Geometry .................................................. 6-2
- 6.2 Discretization ............................................. 6-32
- 6.3 Discrete Stiffeners ..................................... 6-49
- 6.4 Boundary Conditions .................................. 6-57
- 6.5 Loads ...................................................... 6-64
- 6.6 Least-Squares Distributed Line Loads ............. 6-81
- 6.7 Output Control .......................................... 6-88

## 7 Model Input—Element Units (1)
- 7.1 User Points .............................................. 7-2
- 7.2 Line-to-Line–Contact Specifications ............... 7-16

## 8 Model Input—Element Units (2)
- 8.1 Definition of “Spring” Elements via the Edef Protocol ........................................... 8-1
- 8.2 Definition of “Beam” Elements via the Edef Protocol ........................................... 8-9
- 8.3 Definition of Triangle Elements via the Edef Protocol ........................................... 8-14
- 8.4 Definition of “Quadrilateral” Elements via the Edef Protocol ................................. 8-17
- 8.5 Definition of “Other” Elements via the Edef Protocol ........................................... 8-25
  - Contact elements ........................................... 8-28
  - Sandwich elements ....................................... 8-43
  - Solid elements ............................................ 8-103

## 9 Model Input—Element Units (3)
- 9.1 Definition of “Spring” Elements via the Ecom Protocol ........................................... 9-7
- 9.2 Definition of “Beam” Elements via the Ecom Protocol ........................................... 9-16
- 9.3 Definition of Triangular Elements via the Ecom Protocol ...................................... 9-21
- 9.4 Definition of Quad Elements via the Ecom Protocol ............................................ 9-32
- 9.5 Definition of Contact Elements via the Ecom Protocol ........................................ 9-61
- 9.6 Definition of Sandwich Elements via the Ecom Protocol ..................................... 9-73
- 9.7 Definition of Solid Elements via the Ecom Protocol ............................................ 9-131
- 9.8 Definition and Utilization of User Elements ......................................................... 9-163
  - The User–element definition process ................................... 9-163
  - Utilization of User element(s) in **STAGS** .......................... 9-171

## 10 Model Input—Element Units (4)
- 10.1 Element unit loadings .................................. 10-1
- 10.2 Output Control .......................................... 10-14
- 10.3 Linear-Stiffness Contributions ....................... 10-19
# Table of Contents

## 11 Solution Input
11.1 Solution Options ................................................................. 11-1
11.2 Summary and Control Parameters ........................................... 11-4
11.3 The Equivalence Transformation Bifurcation Processor (ET) ............ 11-36
    Simple branch switching .................................................... 11-37

## 12 User-Written Subroutines
12.1 Input/Output Data Conventions ............................................ 12-4
12.2 Subroutine Specifications .................................................. 12-7
12.3 Example Problem ............................................................... 12-66

## 13 User–Defined Elements
13.1 Introduction ........................................................................... 13-1
13.2 User Element Definition Directives ......................................... 13-3
13.3 User Property Set Definition Directives .................................... 13-6
13.4 User Element Model–Definition Routines .................................... 13-9
    Top–level model–definition routines ...................................... 13-9
13.5 User Element Model–Analysis Routines .................................... 13-15
13.6 User Element Post-Processing Routines ................................... 13-20
13.7 FORTRAN– and C–Language Utility Routines ............................ 13-27
13.8 Uniform Beam Example .......................................................... 13-33
    Model- and User-element definition operations .................... 13-35
    User Element Definition Routines .......................................... 13-40
    User beam pre-variation definition routine—UelPvDef900 .......... 13-42
    User beam internal force vector definition routine—UelFiDef900  13-45
    User beam material stiffness matrix definition routine—UelKmDef900 13-49
    User beam strain vector definition routine—UelStrainDef900 ...... 13-53
    User beam stress vector definition routine—UelStressDef900 ...... 13-57
    User beam strain printing definition routine—UelPrintStrainDef900 13-62
    Beam stress printing definition routine—UelPrintStressDef900 .... 13-65
    Analysis and Results ............................................................ 13-67
## 14 The Element Library

14.1 Organization .......................................................... 14-1

14.2 Algorithm for Determining the Element Frame ....................... 14-2

14.3 “Spring” Elements ..................................................... 14-5
   - **E110** Mount element ............................................. 14-6
   - **E120** Rigid link element ........................................... 14-7
   - **E121** Soft link element ............................................ 14-7
   - **E130** Generalized fastener element .............................. 14-8

14.4 “Beam” Elements ..................................................... 14-12
   - **E210** Beam element ............................................... 14-13
   - **E250** Planar boundary condition element ........................ 14-14

14.5 Shell and Mesh-Transition Shell Elements .......................... 14-19
   - **E320** Triangular shell element ................................... 14-20
   - **E330** Triangular shell element ................................... 14-21
   - **E410** 4–Node quadrilateral shell element ....................... 14-22
   - **E411** 4–Node quadrilateral shell element ....................... 14-23
   - **E480** 9–Node quadrilateral shell element ....................... 14-24
   - **E510** and **E710** Quadrilateral mesh-transition shell elements 14-25
   - **E330** Triangular mesh-transition shell elements ............... 14-34

14.6 Sandwich and Mesh-Transition Sandwich Elements .................. 14-36
   - **E830** 6–Node sandwich element .................................. 14-36
   - **E840** 8–Node sandwich element .................................. 14-44
   - **E849** 18–Node sandwich element ................................ 14-45
   - **E845** and **E847** mesh-transition sandwich elements ........ 14-46

14.7 Solid Elements ....................................................... 14-47
   - **E881** 8-Node ANS solid element ................................. 14-47
   - **E882** 18-Node solid element ..................................... 14-48
   - **E883** 27-Node solid element ..................................... 14-49
   - **E885** 20-Node displacement-based solid element ............. 14-50

14.8 Contact Elements ..................................................... 14-51
   - **E810** Pad contact element ....................................... 14-51
   - **E820** General contact element ................................... 14-54
   - **E822** Line contact element ....................................... 14-63
# 15 Analysis Techniques

15.1 Modeling Strategy ........................................ 15-1
15.2 Solution Strategy ........................................ 15-6
   - Linear static analysis ................................ 15-9
   - Eigenvalue analysis .................................. 15-9
   - Nonlinear static analysis ......................... 15-11
   - Transient analysis .................................. 15-17

15.3 Progressive Failure Solution Strategy .................... 15-21
   - Modeling ............................................. 15-21
   - Choosing solution strategy parameters ........... 15-22
   - Maintaining equilibrium ............................ 15-25
   - Recommended guidelines ......................... 15-27

# 16 Interpretation of Results

16.1 Shell Results ........................................... 16-1
16.2 Beam Results ........................................... 16-8
16.3 Evaluating Solution Quality .............................. 16-12
16.4 Interpreting Diagnostic Messages ......................... 16-12
16.5 Overcoming Difficulties ................................ 16-12
   - Linear static analysis .............................. 16-13
   - Eigenvalue analysis ................................ 16-14
   - Nonlinear static analysis ......................... 16-15
   - Transient analysis ................................ 16-17

# 17 Input/Output Files

17.1 Description of I/O Files .................................. 17-1
17.2 Summary of I/O File Requirements ......................... 17-9
   - s1—Model processor .................................. 17-9
   - s2—Solution processor .............................. 17-9
   - STAPL—Plotting processor ......................... 17-9
   - stp—STAGS translator/postprocessor .............. 17-9

# A Appendices

A STAGS Input Record Catalog
1
Introduction

1.1 About this Manual

The “Preface” to this User Manual contains important information about input-data differences between version 5.0 of STAGS and previous versions of the program. The “Using This Documentation” section between the “Preface” and the “Table of Contents” describes methods for navigating within this document and within its companion Elements and Test Cases documents. Some of the key topics covered in this Manual are highlighted in the following Table:

1.2 About STAGS

STAGS is fundamentally a finite element code for general-purpose analysis of shell structures of arbitrary shape and complexity, with additional capabilities for analysis of solids and other types of structural configurations. Shells to be analyzed by STAGS may be thin or thick, unstiffened or stiffened, with stiffeners modeled either as beams or as shells. The availability of numerous wall-fabrication and stiffener-cross-section options combined with a variety of material models permits tremendous flexibility in modeling a wide spectrum of construction types. Analysis capabilities include stress, stability, vibration, and transient analyses, with both material and geometric nonlinearities permissible.

STAGS has a long and distinguished history—starting in the late 1960’s in the Lockheed Missiles and Space Company’s Research and Development Division (R&DD) as a company-sponsored research code to study the stability of shells, and continuing to the present at Lockheed-Martin’s Advanced Technology Center. The primary goal of the STAGS group at Lockheed and at Lockheed-Martin has always been to achieve the best shell stability code available anywhere. Research with and development of STAGS has been under continuous sponsorship from U.S. government agencies and Lockheed’s Independent Research program from the beginning.
## STAGS 5.0—Selected Topics

<table>
<thead>
<tr>
<th>Topic</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installation and execution</td>
<td>Chapter 2 “Installation and Execution”</td>
</tr>
<tr>
<td>Tutorial</td>
<td>Chapter 3 “Getting Started”</td>
</tr>
<tr>
<td>Coordinate systems</td>
<td>Section 4.1 “Coordinate Systems”</td>
</tr>
<tr>
<td>Input data</td>
<td>Chapter 5 “Model Input”</td>
</tr>
<tr>
<td></td>
<td>Chapter 6 “Model Input—Shell Units”</td>
</tr>
<tr>
<td></td>
<td>Chapter 7 “Model Input—Element Units (1)”</td>
</tr>
<tr>
<td></td>
<td>Chapter 8 “Model Input—Element Units (2)”</td>
</tr>
<tr>
<td></td>
<td>Chapter 9 “Model Input—Element Units (3)”</td>
</tr>
<tr>
<td></td>
<td>Chapter 10 “Model Input—Element Units (4)”</td>
</tr>
<tr>
<td></td>
<td>Chapter 11 “Solution Input”</td>
</tr>
<tr>
<td></td>
<td>Appendix B “STAPL Input”</td>
</tr>
<tr>
<td></td>
<td>Appendix L “Design Parameter Derivatives”</td>
</tr>
<tr>
<td>User-written subroutines</td>
<td>Chapter 12 “User-Written Subroutines”</td>
</tr>
<tr>
<td>User-defined elements</td>
<td>Chapter 13 “User–Defined Elements”</td>
</tr>
<tr>
<td>Input and output files</td>
<td>Chapter 17 “Input/Output Files”</td>
</tr>
<tr>
<td>Developing an interface between</td>
<td><strong>STAR</strong> Reference Manual</td>
</tr>
<tr>
<td>STAGS and a pre- or post-processor</td>
<td></td>
</tr>
</tbody>
</table>

Today, STAGS has a world-wide distribution and is used extensively throughout government, industry and academia.

### Analysis and structure types

STAGS has the following analysis capabilities:
STAGS is used extensively throughout the aerospace, shipbuilding and other industries for analysis of panels, pressure vessels, and general shell structures. While it can be efficiently applied to routine linear systems, STAGS’ forte is the analysis of complex, nonlinear systems that depend on post-buckling strength and require analysis well into the post-buckled regime. STAGS is routinely used for pre- and post-test verifications of complex systems—especially those that are sensitive to initial geometric imperfections, which can be defined in STAGS with ease and flexibility.

STAGS has been used by NASA for Space Shuttle and Aging Aircraft analyses. It was used to perform a buckling analysis of the pear-shaped orbiter fuselage, and it was used for analysis of the solid rocket booster during the redesign activities following the Challenger event. It has been used for analysis of advanced aircraft, such as the High Speed Civil Transport (HSCT), and is currently being used for analyses of crack propagation, delaminations, and other phenomena that occur in aging aircraft structures.

STAGS has played a major role in pressure-hull design of U.S. Navy submarines, and it has been coupled with USA (Underwater Shock Analysis) in the USA-STAGS code for the shock-load analysis of submarine structures.

STAGS has been used for analysis of nuclear reactor containment vessels, strategic missiles, and advanced solid rocket motor cases. It is also used for the analysis of almost every structure designed in the Lockheed’s RDD for cryogenic coolers, space experiments, and satellite systems such as antennas, mirrors, and telescopes.

Following is a summary of some of the structure types for which STAGS is applicable:

- linear elastic stress analysis
- geometrically nonlinear elastic stress analysis
- geometrically linear inelastic stress analysis
- geometrically nonlinear inelastic stress analysis
- linear bifurcation buckling analysis based upon either a linear or a nonlinear stress state
- branch on bifurcation from a primary path to a secondary solution path
- linear vibration analysis based upon either a linear or a nonlinear stress state
- transient response analysis, elastic or inelastic, geometrically linear or nonlinear
Introduction

STAGS’ capabilities for solution of difficult, nonlinear shell problems are typified by Figure 1.1 through Figure 1.4, which show examples of some of the kinds of problems for which STAGS is applicable.

- panels
- pressure vessels
- aircraft
- spacecraft
- offshore oil platforms
- ships
- submarines
- cryogenic coolers
- reactor containment vessels
- solid rocket motor cases
- satellite systems
- space experiments
- strategic missiles
- general shell structures
Solution algorithms

Solution control in STAGS is quite sophisticated, ranging from simple load control to the advanced Riks arc-length parameter technique that enables traversal of limit points into the post-buckling regime, to equivalence transformation methods to obtain solutions during mode-jumping behavior that may occur in bifurcation and post-buckling response. For arc-length algorithms, the solution can be directed to stop at maximum load, or to go into a negative direction for determination of permanent set. Two load systems with different histories can be defined at the same time and can be controlled separately during the solution process.

STAGS solutions can be saved according to user instructions for restart or for postprocessing. The state at termination is always saved for future use. Flexible restart procedures permit switching from one strategy to another during an analysis; this includes a shift from static to transient, or from transient to static with modified boundary conditions and loadings.

Figure 1.2  Cylindrical panel with circular cutout. Left: undeformed shape. Right: buckled shape due to axial compression.
STAGS provides solutions to the generalized eigenvalue problem for buckling and vibration from a linear or nonlinear stress state. A robust solution algorithm allows for efficient shifting to the neighborhood of eigenvector-eigenvalue pairs sought.

Figure 1.3  One-quadrant model of a jet aircraft engine support ring. Above: undeformed shape. Below: deformation due to a severe thermal gradient, plasticity, and creep.
An element-independent, large rotation algorithm is in place, allowing for analysis of structures undergoing arbitrary displacements and moderate strains, with no artificial stiffening due to large rotations. Not only are all elements objective to arbitrary rigid-body motion, but the first and second variations of the strain energy are consistent, resulting in quadratic convergence for true-Newton iteration sequences.

**STAGS** can also treat some types of structural contact problems—with very efficient element-on-element (Hertzian) PAD-type contact elements when the contact region is known *a priori*, with more general point/surface contact elements when the contact region is not known *a priori* and/or when element-on-element contact is not expected and/or slippage occurs, and with line and line-interaction contact elements.

**Modeling**

Quadratic surfaces can be modeled automatically in **STAGS**, with minimal user input, as individual substructures, called *shell units*. An important feature here is that the analytic geometry is represented exactly—there is no need to be concerned with accuracy limitations imposed by parametric representation, such as that typically found in modern finite-element modeling software. These shell units can be generated and interconnected along edges, along

**Figure 1.4** Complex titanium pressure vessel. Left: undeformed shape. Right: plastic collapse analysis under hydrostatic pressure.
internal grid lines, or in more general ways, with provisions in the program for treating either partial or complete compatibility. Using these techniques, complex structures can be assembled. In addition to these automatically-generated shell units describing quadratic surfaces, *element units* may be defined as arbitrary assemblages of nodes and elements. Element units may be used to complement a structure that is made up of STAGS-generated shell units and/or to form a self-contained structural model generated by STAGS and/or by other programs.

**Material, wall & stiffener properties**

A variety of material models is available, including both plasticity and creep. This, with numerous wall-fabrication and stiffener-cross-section options, permits tremendous flexibility in modeling a wide spectrum of construction types. STAGS handles isotropic and anisotropic materials, including composites consisting of up to 100 layers of arbitrary orientation. Four plasticity models are available, including the often used isotropic strain hardening, White-Besseling (mechanical sublayer model), kinematic strain hardening, and deformation theory. Progressive failure analysis models are available for laminated composite structures using five common failure criteria and material degradation models. Easily defined standard wall types include orthotropic laminate, fiber-wound, and corrugation-stiffened. Additional generality is provided by the option for direct input of stiffnesses relating stress and moment resultants to surface strains and curvatures. Stiffener cross sections may be built-up from subelements described either by explicit geometry or by geometric properties. Smeared stiffeners are also an option.

**Loads**

Two independent load sets, each composed from simple parts that are easily specified with minimal input, define a spatial variation of loading. Any number of prescribed displacements, point loads, line loads, surface tractions, thermal loads, and “live” pressure loads (hydrostatic pressure that remains normal to the shell surface throughout large deformations) can be combined in this manner to make a load set. Inertial loadings can be specified for the structure as well. For transient analysis, the user may select from a menu of convenient loading histories; or he/she may specify a completely-general temporal variation via a user-written subroutine.

**Boundary conditions**

Boundary conditions (BCs) may be imposed either by reference to certain standard conditions or by the use of single-point and multi-point constraints. Simple support, symmetry, antisymmetry,
clamped, or user-defined BCs can be specified on a shell unit edge. Single-point constraints which allow individual freedoms to be free, fixed, or a prescribed nonzero value may be applied to grid lines and surfaces in shell units. A useful feature for buckling analysis allows these constraints to differ for the prestress and eigenvalue analyses. Lagrangian constraint equations containing up to 100 terms may be easily defined to impose multi-point constraints. Advanced features include a “moving-plane” boundary permitting arbitrary motion of a boundary while requiring all nodes on the boundary to remain coplanar.

Element library

STAGS has a variety of finite elements that are suitable for the analysis of stiffened plates, shells, and solids.

Simple 4-node quadrilateral plate elements with a cubic lateral displacement field provide a very effective and efficient building block for the description of complex post-buckling thin-shell response. A linear or quadratic membrane interpolation can be selected. For thicker shells in which transverse shear deformation is important, STAGS provides the Assumed Natural Strain (ANS) 9-node element which has proven to be a sturdy performer for complex structures. A 2-node beam element that is compatible with the 4-node quadrilateral plate element is provided to simulate stiffeners and beam assemblies. For irregular geometries, Clough-Tocher triangular plates can be combined automatically to make quadrilaterals. In addition to these standard shell elements, there are two mesh-transition shell elements that can be used very effectively with shell structures in which there are transitions from finer to coarser grids. One of these is used along an edge that has 2N shell elements on one side and N shell elements along the other (with two elements on the first side paired with a single element on the second). The other is used at junctures where two (or three) finely-meshed shell regions meet a coarsely-meshed region at a single point. Each of these elements is capable of describing large-displacement, moderate-strain response because the element-independent large rotation algorithm, described previously, functions with all types of finite elements.

STAGS has four standard solid elements that are suitable for treating thick shells and 3-D solid structures. The simplest of these elements—the 8-node brick—can be used very effectively in structural regions that have minimal geometric and material variations and that experience reasonably uniform or slowly varying responses. The other three solid elements—which have 18, 20 and 27 nodes—are suitable for structural regions in which geometric, material and structural characteristics and responses are nonuniform in any or all of these domains.

STAGS has a two standard elements that are suitable for treating sandwich shell structures. The simpler of these elements—the 8-node sandwich—uses two 4-node quadrilateral shell elements to model the upper and lower surfaces (one each) and one or more 8-node bricks to model the
solid core between the upper and lower surfaces of the element. This hybrid element can be used most effectively in structural regions that have minimal geometric and material variations and that experience reasonably uniform or slowly varying responses. The more refined—18-node sandwich—element uses two 9-node quadrilateral shell elements to model the upper and lower surfaces and one or more 18-node solid elements to model the core. This element is most useful in structural regions where geometric, material and structural characteristics and responses are more complex. In addition to these two standard sandwich elements, there are two mesh-transition sandwich elements that can be used very effectively with structures in which there are transitions from finer to coarser sandwich-element grids. One of these is used along an edge that has 2N sandwich elements on one side and N sandwich elements along the other (with two elements on the first side paired with a single element on the second). The other is used at junctures where two (or three) finely-meshed sandwich regions meet a coarsely-meshed sandwich region at a single point.

In addition to these standard and hybrid finite elements, STAGS provides several types of specialized elements. For transition from a finer to a coarser grid, there is a line-constraint element that enforces edge compatibility of the fine mesh with the coarse mesh. The nonlinear mount element is a generalized spring with a user-input force-displacement-velocity profile that can serve as a connector, with offset, between any two nodes in the system. By a suitable choice of input, the user can model very complex mount response, including nonlinear damping. Other specialized elements are involved in the moving-plane boundary condition described previously.

**Initial geometric imperfections**

Initial imperfections can be defined automatically in a variety of ways, permitting imperfection-sensitivity studies to be performed with minimal effort. This can be an invaluable aid in numerous situations, such as when analyzing a weight-critical design where there is uncertainty about knockdown factors for buckling, and/or for determining allowable manufacturing tolerances. Imperfections can be generated from linear combinations of previously-computed buckling modes; from a trigonometric expansion using a simple input of mode numbers, shell-surface locations, and amplitudes; or from user-written subroutines. A special random imperfection capability allows for randomly-generated amplitudes with specified standard deviation and mean values. Combinations of buckling modes and trigonometric data are permitted. A higher-order definition of the imperfection waveform is used in element computations.
User-written subroutines

The option of using user-written subroutines is a very powerful feature of STAGS. User-written subroutines can be used in very basic ways as elegant alternatives to lengthy input decks, or in more sophisticated ways to extend the generality of the code. Users can easily define arbitrary functional relationships, such as spatially-varying geometric and constitutive properties in a wall fabrication, or general time-varying pressure loads. Nearly all aspects of model definition, from the most basic to extremely complex ones, can be performed via user-written subroutines. See Chapter 12 for more about this feature.

User-defined elements

The current version of STAGS gives the user the very powerful capability to install and employ User-defined elements. See Chapter 13 for more about this feature.
Methods for installation, maintenance and execution of STAGS programs on various computer platforms are described in this chapter. Before getting into that, however, a few words about how the STAGS program system is organized and about how this chapter is organized are in order.

One important thing to note here is that the STAGS program system “looks” the same (i.e., it has the same software components and is organized in the directory structure) on all of the (UNIX, Linux, Macintosh and Windows-based) computer platforms on which it operates. The directory structure that STAGS uses on all of these platforms is described in Section 2.1 of this chapter. The “administrator” who installs STAGS on any given machine should read Section 2.1 because he (or she) must understand this directory structure and operate within it. Anyone who uses STAGS on any installation needs to understand this directory structure, too. This is especially true for users who want or need to construct and execute customized versions of STAGS processors utilizing user-written subroutines, user-defined elements, or any other non-standard components.

Methods for installing, maintaining and executing STAGS are discussed next. Computer requirements, installation procedures, maintenance methods and execution procedures for STAGS are generally computer-system and computing-environment dependent. These four subjects are discussed in a separate section of this chapter for each of the currently supported platforms:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>2.2 “UNIX-Based Systems” on page 2-6</td>
</tr>
<tr>
<td>Linux</td>
<td>2.3 “Linux-Based Systems” on page 2-17</td>
</tr>
<tr>
<td>Macintosh</td>
<td>2.4 “Macintosh-Based Systems” on page 2-26</td>
</tr>
<tr>
<td>Windows</td>
<td>2.5 “Windows-Based Systems” on page 2-35</td>
</tr>
</tbody>
</table>

In each of these four platform-specific sections, there are three subsections: the first contains a concise summary of the basic system requirements (operating systems supported, free disk space needed, utilities and compilers needed, etc.) for installing and executing STAGS programs on the
platform in question; the second has important information that the administrator who installs STAGS on that platform needs; and the third has vital information that anyone who executes STAGS programs on that platform needs. STAGS administrators should read the “Basic... System Requirements” and the “Installation...” subsections for the platform onto which STAGS is to be installed. All users should read the “Execution...” part of the section for the platform on which STAGS is to be executed.

The final section of this chapter (“Installation Verification” on page 2-36) contains information that the STAGS administrator needs to validate the installation and performance of STAGS on any of these platforms. This information may be of interest and value to other STAGS users, as well.
2.1 STAGS Directory Structure (all platforms)

The most important components of the STAGS directory structure (on all platforms) are shown schematically in Figure 2.1:

![STAGS directory structure (all platforms)](image)

**Figure 2.1** STAGS directory structure (all platforms)
Symbols, conventions and meanings of graphic and text elements in Figure 2.1 are explained in Figure 2.2:

![Diagram of STAGS directory structure]

**Figure 2.2** Symbols used in the STAGS directory diagram.

With one exception, each (yellow) $STAGSHOME/lib$ library directory contains three files:

- **makefile@**: this is a soft link to the makefile (makefile.lib, in the “make” directory) that is actually used to build the desired library file lib.a for this particular library directory; only one of these soft links (for the cu library) is shown (as a blue, arrow-tipped line) in Figure 2.1

- **src@**: this is a soft link to the (green) subdirectory (src/lib) that contains the FORTRAN or C-language source code that is required to build the library file for this particular library directory; only one of these soft links (for the cu library) is shown (as a blue, arrow-tipped line) in Figure 2.1

- **lib.a**: this is the desired library file for this particular library directory

The sole exception to this pattern is the $STAGSHOME/vss$ library directory. Source code for the vss equation solver is proprietary and is not typically distributed with the STAGS program, so the $STAGSHOME/vss$ library directory only contains the pre-built library file vss.a for the platform to be used.

When STAGS is installed correctly, the $STAGSHOME/bin$ directory (see Figure 2.1) contains seven executable STAGS processors (s1, s2, stap, scopy, xytrans, pitrans and pat2s) and one soft-link file (makefile). The makefile soft link (a blue, arrow-tipped line, in Figure 2.1)
“points” to the $STAGSHOME/make/makefile.all: this is a makefile script that is used to “make” all of the libraries and programs in the STAGS system.

When STAGS is installed correctly, the $STAGSHOME/prc directory contains a number of script and other files that are used to establish a standardized STAGS environment, to run STAGS programs, and to manage STAGS resources and STAGS-related user files. These processors, links, scripts and files are discussed in the following subsections.
2.2 UNIX-Based Systems

2.2.1 Basic UNIX System Requirements

Basic system requirements that must be met to install STAGS on a UNIX system are summarized in the following table and text:

<table>
<thead>
<tr>
<th>Machine</th>
<th>Type</th>
<th>OS</th>
<th>Disk Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC</td>
<td>Alpha workstation</td>
<td>??</td>
<td>??</td>
</tr>
<tr>
<td>HP</td>
<td>32-bit architecture workstation</td>
<td>HP-UX</td>
<td>80 MB</td>
</tr>
<tr>
<td>HP</td>
<td>64-bit architecture &amp; code, 4-byte integers</td>
<td>HP-UX</td>
<td>80 MB</td>
</tr>
<tr>
<td>HP</td>
<td>64-bit architecture &amp; code, 8-byte integers</td>
<td>HP-UX</td>
<td>80 MB</td>
</tr>
<tr>
<td>IBM</td>
<td>RS_6000 workstation</td>
<td>??</td>
<td>??</td>
</tr>
<tr>
<td>SGI</td>
<td>32-bit IRIS workstation</td>
<td>??</td>
<td>140 MB</td>
</tr>
<tr>
<td>SGI</td>
<td>64-bit architecture, 32-bit object code</td>
<td>??</td>
<td>140 MB</td>
</tr>
<tr>
<td>SGI</td>
<td>64-bit architecture &amp; code, 4-byte integers</td>
<td>??</td>
<td>140 MB</td>
</tr>
<tr>
<td>SGI</td>
<td>64-bit architecture &amp; code, 8-byte integers</td>
<td>??</td>
<td>140 MB</td>
</tr>
<tr>
<td>SUN</td>
<td>SUN-4 / SPARC workstation</td>
<td>Solaris</td>
<td>85 MB</td>
</tr>
<tr>
<td>SUN</td>
<td>SPARC workstation</td>
<td>SunOS</td>
<td>85 MB</td>
</tr>
</tbody>
</table>

Compilers required: FORTRAN & C-languages

Other system requirements that must be met (and should be comfortably exceeded) to execute STAGS programs on a UNIX system are summarized in the following table:

- Processor speed requirements: 300 MHz
- Memory requirements: 256 MB
- Storage requirements: 512 MB

2.2.2 Installation of STAGS on a UNIX System

On a UNIX system, one must be running the C-Shell to install STAGS. If the C-Shell is not your default shell, you must change to csh to perform the installation.
Installing STAGS on a UNIX system

UNIX versions of STAGS are typically distributed on CDs or via “ftp” as compressed tar or gzip-formatted archive files. The first step in installing STAGS on a UNIX system is to transfer the contents of the distribution media onto the computer system to be used.

When STAGS is distributed as a compressed tar archive file, the name of the file is typically $\text{stags}_{\text{type}}.\text{tar.Z}$, where $\text{type}$ is an abbreviation for the type of machine onto which STAGS is to be installed. In this case,

1) Move the compressed tar file to the directory in which you want the stags directory to reside (into $\text{/usr/local}$, for example).

2) Issue the following command, to decompress the compressed tar file and construct the desired stags directory:

```
% zcat stags_{type}.tar | tar xvf -
```

When STAGS is distributed as a gzip-formatted archive file, the name of the file is typically $\text{stags}_{\text{type}}.\text{tgz}$, where $\text{type}$ is an abbreviation for the type of machine onto which STAGS is to be installed. In this case,

1) Move the gzip-compressed archive file to the directory in which you want the stags directory to reside (into $\text{/usr/local}$, for example).

2) Issue the following command, to unzip the compressed archive file into its tar-formatted form:

```
% gunzip stags_{type}.tgz
```

3) Issue the following command, to unwrap the tar-formatted file and construct the desired stags directory:

```
% tar xf stags_{type}.tar
```

Having unloaded (and decompressed) the archive file, you should then have a new subdirectory, named stags, that is installed in the current working directory. This directory is referred to as $\$\text{STAGSHOME}$ in the file directory diagram that is shown in Figure 2.1. Continue the installation process by executing the RUN.ME.FIRST script from the new stags subdirectory, as follows:

```
% cd stags
% RUN.ME.FIRST
```

The RUN.ME.FIRST script modifies certain STAGS files to complete the installation, then it commits suicide by deleting itself. Installation is now complete. Before STAGS can be executed, however, it is necessary to perform an initialization step that establishes a number of command aliases and environmental variables that are needed to operate on and with STAGS programs.
Initializing STAGS on a UNIX system

Issue the following command once per login session prior to changing or running STAGS:

```bash
% source $STAGSHOME/prc/initialize
```

where $STAGSHOME is the complete path name to the stags directory structure that is shown in Figure 2.1. The $STAGSHOME/prc/initialize procedure establishes a number of command aliases and environmental variables that are necessary to operate on and/or to execute the STAGS programs. For example, $STAGSHOME is defined as an environmental variable that contains the path name of the stags directory.

It is strongly recommended that each STAGS user inspect the $STAGSHOME/prc/initialize file to see the aliases and variables that it creates, and install this command (or define an alias for it) in his or her .login file.

Making STAGS on a UNIX system (as and if necessary)

For UNIX installations, STAGS is typically distributed with source code and with object libraries and executables, for specific machine architectures. Normally, it is not necessary for the administrator who installs STAGS or for the typical STAGS user to “make” STAGS programs. In unusual circumstances—such as when inconsistencies exist between the environment used to create the STAGS distribution software and the environment on the user’s machine—it may be necessary for the administrator (and/or for the user) to do so. The computationally-sophisticated user may elect to recompile and re-link STAGS components to take advantage of high-performance compilation and/or optimization features that may be available on his or her machine.

To make STAGS, simply execute the UNIX make command from the $STAGSHOME directory. To make specific libraries or executables, issue the following command from any directory:

```bash
% makestags [target]
```

See the file $STAGSHOME/prc/makestags for a list of valid targets. To remake the star library, for example, execute the command:

```bash
% makestags star.a
```

Invocation of the makestags command, with no arguments, will make all of STAGS. It is equivalent to executing the make command from $STAGSHOME.
Coping with installation errors (as and if necessary)

Some common causes of installation difficulties include insufficient disk space, lack of write permission in the selected directory, attempting to perform installation from a shell other than the C-shell, nonexistence of UNIX commands that are required for the installation procedure, and installer boneheadedness. If installation is not successful, identify and correct the problem; then delete the entire $STAGSHOME directory and start over again at the 2.2.2 “Installation of STAGS on a UNIX System” point.

Verifying that STAGS operates correctly

Correct operation of STAGS can be verified by running the small suite of test cases in the $STAGSHOME/testcases directory and examining selected portions of the output. Basic concepts and procedures for executing STAGS processors on UNIX systems are described in the following subsection—which the administrator who installs STAGS and anyone who uses it on such a system must master. A first-level operation verification procedure for STAGS is discussed in Section 2.6 (at the end of this chapter).
2.2.3 Execution of STAGS on a UNIX System

Before we address this subject, it is important for the reader to understand that there are two basic types of executable STAGS programs:

- **default executables** these are the standard versions of the STAGS processors (s1, s2, stapl, scopy and xytrans) that are supplied with the program and used for most applications; these executables are stored in the $STAGSHOME/bin directory
- **custom executables** these are application-dependent versions of s1, s2, stapl, scopy and/or xytrans—which are typically created by linking with user-written subroutines and/or user-defined elements and are typically stored in user-controlled directories.

On UNIX systems, the $STAGSHOME/prc directory contains a number of procedure files that facilitate the execution and construction of STAGS programs. These procedure files (scripts) are described in the $STAGSHOME/README file. The most important of these (in the “execution” department) is the stags procedure—which is typically used to run

- s1 STAGS’ model processor
- s2 STAGS’ solution processor
- stapl STAGS’ plotting post-processor
- scopy STAGS’ file-copy processor
- pitrans STAGS’ translator/post-processor
- xytrans STAGS’ post-processor to generate data for XY plots

to construct a STAGS model, to perform one or more analyses with it, to perform pre- and/or post-analysis plotting operations, and to do other things with data on STAGS I/O files.* The most important of these procedures (in the “construction” department) is makeuser—which is used to “make” custom versions of STAGS processors when it is necessary to do so. The remainder of this subsection is devoted to descriptions of these two procedures:

```text
stags:
```

On UNIX systems, one typically “runs” the STAGS program(s) by invoking the $STAGSHOME/prc/stags procedure; this is typically done via the stags alias, which is constructed by $STAGSHOME/prc/initialize when the user initializes STAGS prior to exercising it; the stags procedure can be used to run STAGS with default and/or with custom STAGS executables.

See “Executing STAGS with stags (UNIX)” on page 2-12 for information about the stags procedure, and “Initializing STAGS on a UNIX system” on page 2-8 for more information about the initialization process.

* See Chapter 17 “Input/Output Files” for documentation on STAGS I/O files.
Custom, user-owned versions of executable STAGS processors may be created with the $STAGSHOME/prc/makeuser procedure, which is generally executed via the makeuser alias that $STAGSHOME/prc/initialize defines when the user initializes STAGS prior to exercising it; custom executables must be created and executed when user-written subroutines and/or user-defined elements are employed.

See “Creating custom STAGS executables with makeuser (UNIX)” on page 2-14 for more information about the makeuser procedure.

See “Initializing STAGS on a UNIX system” on page 2-8 for information about the initialization process.

See Chapter 12 “User-Written Subroutines” for more information about user-written subroutines, and Chapter 13 “User–Defined Elements” for more information about user-defined elements.

Users can also create special-purpose versions of STAGS code, since source code is distributed with the program; this type of advanced use is outside the scope of this document, but the sophisticated analyst may find occasion to customize STAGS for specific applications.
Executing STAGS with stags (UNIX)

Synopsis

    stags [options] casename [options]

Options

- **b** – run STAGS in background; the default nice value is 10.
- **n** [nice_value] – set nice value for non-queued jobs; if option -n is selected, but a nice value is not supplied, the default is 10.
- **sm** – send email message upon termination of background jobs; default is send no email; queued jobs always send email upon termination.
- **q** – submit STAGS job to NQS (Network Queuing System) queue.
- **t** seconds – set time-limit for queued job to seconds; the default is 900.
- **1** [s1_executable] – run s1; a user-specified s1 executable is optionally defined.
- **2** [s2_executable] – run s2; a user-specified s2 executable is optionally defined.
- **rh** [remote_hostname] – copy files from specified hostname into local $TMPDIR for execution and then put the results back on the remote host upon completion.
  remote_hostname must appear in user’s ~/.netrc file; primarily intended for use on highly-impacted systems such as CRAYs.
- **rd** [remote_directory] – directory on remote machine from which to copy input data files and to which to copy output files; valid only when remote_hostname is specified; default is ~.

Examples

- run the default s1 and s2 executables interactively or in the foreground
  % stags casename
  If neither -1 nor -2 is present, then both s1 and s2 will be run in succession, using the default executables.

- run a custom version of s2 with the default version of s1
  % stags -1 -2 [s2_executable] casename

- run a custom version of s1 with the default version of s2
  % stags -2 -1 [s1_executable] casename

- run default s1 and custom s2 in background, then send email upon completion
  % stags -b -1 -2 ~/my_s2 -sm mycase

- run custom s2 in the background, then send mail upon completion
  % stags -b -sm -2 ~/my_s2 mycase

- run s1 in the foreground
  % stags mycase -1
Note that options -1/-2 appear after mycase in the above/below examples. Any argument following -1 or -2 must be either a valid STAGS executable (custom or default) or another option (i.e., an argument beginning with a -).

- run s2 in the foreground
  
  \% stags mycase -2

- NQS queuing system
  
  \% stags -q -t 1000 mycase

  The above command will submit the STAGS job using mycase as input to the queue that runs jobs for at least 1000 seconds. Both the default s1 and s2 will be executed.

- remote get & put of I/O files
  
  \% stags -rh trinity.rdd -rd work/stags mycase -q

  On some machines (often CRAYs), available user disk space may be at a premium; users may wish to store input and output data on their local machines and instruct STAGS to get the input files and put the output files using ftp; the above command will fetch input files for mycase from directory ~/work/stags on remote host trinity.rdd, submit the job to the NQS queue of the current host computer, and copy the output files back to trinity.rdd; diagnostics will always be emailed when using the remote host option because the “mycase.log” file always gets left behind on the executing machine; users should be aware that previous generations of output files are not retained when using this option; the warning that the input files cannot be found on the current host may be ignored.
Creating custom STAGS executables with makeuser (UNIX)

Synopsis

```
makeuser [-g] [-l userlib.a] [target]
```

Options

- `-g`  compile .F files with standard debug option
- `-cpp "cppflags"`  where `cppflags` are valid `cpp` options; user-entered `cppflags` are both prepended and appended to the default `cppflags`; the user-entered `cppflags` are prepended so that, if given, alternate include-file directories can be searched before the standard include-file directories; the user-entered `cppflags` are also appended so that default `cpp` definitions can be reset by the user; the user-entered `cppflags` do not replace the default `cppflags` because STAGS code will not compile correctly without certain default `cpp` definitions that the user is likely to forget to include.  
Example:  `-cpp "-D_debug_"`
- `-fc "fcflags"`  where user-entered `fcflags` (FORTRAN compiler options) replace the default compiler options for all modes of compilation: no optimization, partial optimization, full optimization, and debug; on most machines, users should remember to include the “-c” flag if they exercise this option.
Example:  `-fc "-c -g -O2"`
- `-cc "ccflags"`  where user-entered `ccflags` (C compiler options) replace the default compiler options.
Example:  `-cc "-c -g -O2"`
- `-ld "linkflags"`  where user-entered `linkflags` (linker options) replace the default linker options.
Example:  `-ld "-M -Bstatic"
- `-l userlib.a`  link library `userlib.a` when target is made; more than one library can be linked using additional “-l” arguments.

`target` is a target in `$STAGSHOME/make/makefile.user`. Valid targets are:

- `s1`  creates a custom version of `s1`, the STAGS model processor
- `s2`  creates a custom version of `s2`, the STAGS solution processor
- `scopy`  creates a custom version of `scopy`, a STAGS data postprocessor
- `pitrans`  creates a custom version of `pitrans`, a STAGS translator/postprocessor
- `xytrans`  creates a custom version of `xytrans`, a STAGS xy-plot-data translator
- `clean`  removes custom object and executable files from the current working directory

If no `target` is specified, then `makeuser` attempts to make all STAGS executables, placing them in the current working directory (see Table 2.1).
Each user-written subroutine must be in a .F source file. Routines called by user-written subroutines, if any, can be in .c files, .F files, and/or in .a library files; do not duplicate any STAGS names.

The `makeuser` procedure compiles source files (.F, .c, and .h, as appropriate) in the current working directory and links these files ahead of the standard STAGS libraries. If user libraries are included, they will be linked after any source files in the current directory but before any STAGS libraries.

The `makeuser` procedure operates exclusively on files that are in the current working directory ($cwd) and has no effect on the $STAGSHOME file system or on any other file system.

<table>
<thead>
<tr>
<th>target</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>All of the source files (.F, .c, and .h) in the current directory are compiled and the object files produced by those compilations are placed in the user.a library. All of the s1 program files in user.a are extracted and linked ahead of all other libraries to create the us1 executable.</td>
</tr>
<tr>
<td>s2</td>
<td>All of the source files (.F, .c, and .h) in the current directory are compiled and the object files produced by those compilations are placed in the user.a library. All of the s2 program files in user.a are extracted and linked ahead of all other libraries to create the us2 executable.</td>
</tr>
<tr>
<td>stapl</td>
<td>All of the source files (.F, .c, and .h) in the current directory are compiled and the object files produced by those compilations are placed in the user.a library. All of the stapl program files in user.a are extracted and linked ahead of all other libraries to create the ustapl executable.</td>
</tr>
<tr>
<td>scopy</td>
<td>All of the source files (.F, .c, and .h) in the current directory are compiled and the object files produced by those compilations are placed in the user.a library. All of the files in user.a are extracted and linked ahead of all other libraries to create the scopy, pitrans and xytrans executables.</td>
</tr>
<tr>
<td>pitrans</td>
<td>Any of the following files which exist in the current working directory are removed:</td>
</tr>
<tr>
<td>xytrans</td>
<td>_*</td>
</tr>
<tr>
<td>clean</td>
<td>*.o</td>
</tr>
<tr>
<td></td>
<td>any STAGS executable</td>
</tr>
<tr>
<td></td>
<td>user.a</td>
</tr>
</tbody>
</table>
CAUTION: Existing .o files are erased by makeuser. These may be placed in a .a library file, which may be linked with the “-l” option.

makeuser attempts to “force link” any object modules produced from source files, and some linkers will abort if asked to force link a name that is not referenced.; this vexation can be avoided by ensuring that $cwd includes only those source files containing program modules whose names are referenced by the target.

Examples

Suppose the current working directory contains the following files:

```
-rw-r--r--  1 stags       12833 Jan 12 10:30 mylib1.a
-rw-r--r--  1 stags       78294 Jan 12 10:30 mylib2.a
-rw-r--r--  1 stags        1242 Jan 12 10:30 user2.F
-rw-r--r--  1 stags        7231 Jan 12 10:30 user1.F
```

- create a custom version of s2, incorporating user1.F and user2.F
  ```
  % makeuser s2
  ```

- create a custom version of s2—with user1.F and user2.F compiled for symbolic debugging
  ```
  % makeuser -g s2
  ```

- create a custom version of s2—linking in mylib1.a and mylib2.a, in addition to user1.F and user2.F
  ```
  % makeuser -l mylib1.a -l mylib2.a s2
  ```

- create a custom version of s2—linking in mylib1.a and mylib2.a, together with user1.F and user2.F compiled for symbolic debugging:
  ```
  % makeuser -g -l mylib1.a -l mylib2.a s2
  ```

In all of these examples, makeuser will produce a user.a user–library file and a us2 user–executable file. Since no .c files and no .h files exist in the current directory, makeuser will also create dummy files dummyc.c and dummyi.h.
2.3 Linux-Based Systems

PRELIMINARY:

The good news here is that STAGS can be installed easily and can be executed effectively on various flavors of Linux-based machines. The bad news here is that this seriously underfunded section of Chapter 2 is still “under construction” and is somewhat incomplete, right now. While this is the case, we recommend that anyone who experiences difficulties in installing and executing STAGS on a Linux-based system contact contact Dr. Charles C. Rankin, at

Rhombus Consultants Group
2565 Leghorn Street
Mountain View, CA 94043

or Dr. Frank C. Weiler, at

Lockheed-Martin Missiles & Space Co., Inc.
ATC Org. L9-21; Bldg. 204
3251 Hanover Street
Palo Alto, CA 94304

2.3.1 Basic Linux System Requirements

Basic system requirements that must be met to install STAGS on a Linux system are summarized in the following:

- Minimum disk space required: 85 MB
- System utilities required: fpp & fsplit (supplied)
- Compilers required: gcc & g77

System requirements that must be met (and should be comfortably exceeded) to execute STAGS programs on a Linux system are summarized in the following table:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor speed requirements</td>
<td>300 MHz</td>
</tr>
<tr>
<td>Memory requirements</td>
<td>256 MB</td>
</tr>
<tr>
<td>Storage requirements</td>
<td>512 MB</td>
</tr>
</tbody>
</table>

2.3.2 Installation of STAGS on a Linux System

STAGS is conveniently installed on Linux systems while operating within the framework of a C-Shell environment.
Installing STAGS on a Linux system

Linux versions of STAGS are typically distributed on CDs or via “ftp” as a pair of gzip-compressed archive files—the primary distribution file (stags_linux.tgz) and a secondary (but necessary) utility file (ftp_split.tgz). Installation of STAGS on a Linux system is typically a six-step process:

1) Establish the root directory into which STAGS is to be installed (the /home/stags directory, for example)—creating it as and if necessary; make this directory the current working directory.

2) Copy the two STAGS distribution files from their distribution medium into this working directory.

3) Extract STAGS from the stags_linux.tgz file and the fpp and fsplit utilities from the fpp_fsplit.tgz file by issuing the following two commands:

   % tar xzf stags_linux.tgz
   % tar xzf fpp_fsplit.tgz

   The first operation in this step creates a new subdirectory (named stags) within the current working directory. This new subdirectory is called $STAGSHOME in the directory structure diagram that is shown in Figure 2.1. After creating this new subdirectory, the first operation constructs the STAGS program system within it.

   The second operation in this step creates two additional subdirectories (named fpp and fsplit) within the current working directory. The fpp and fsplit subdirectories contain two utility programs that are needed whenever it is necessary to compile STAGS routines and/or to “make” STAGS executables.

4) Continue the installation process by executing the RUN.ME.FIRST script from the new stags subdirectory, as follows:

   % cd stags
   % RUN.ME.FIRST

   The RUN.ME.FIRST script modifies the STAGS initialization file (discussed next); then it commits suicide by deleting itself.

5) Continue the installation process by transferring from the stags subdirectory to fpp (one of stags’ two siblings) and performing the following operations:

   % cd ../fpp
   % make fpp

   The make fpp command compiles and “makes” the fpp utility program, producing fpp*. Transfer (copy or move) fpp* from the current working directory into the /usr/local/bin directory (which must be in the user’s path).
6) Conclude the installation process by transferring from the fpp subdirectory to fsplit (stags' other sibling) and performing the following operations:

% cd ../fsplit
% make fsplit

The make fsplit command compiles and “makes” the fsplit utility program, producing fsplit*. Transfer (copy or move) fsplit* from the current working directory into the /usr/local/bin directory (which must be in the user’s path).

Installation is now complete. An initialization procedure must be performed, however, before STAGS can be executed.

Initializing STAGS on a Linux system

Issue the following command once per login session prior to changing or running STAGS:

% source $STAGSHOME/prc/initialize

where $STAGSHOME is the complete path name to the stags directory structure that is shown in Figure 2.1. The STAGSHOME/prc/initialize procedure establishes a number of command aliases and environmental variables that are necessary to operate on and/or to execute the STAGS programs. For example, $STAGSHOME is defined as an environmental variable that contains the path name of the stags directory.

It is strongly recommended that each STAGS user inspect the $STAGSHOME/prc/initialize file to see the aliases and variables that it creates, and install this command (or define an alias for it) in his or her .login file.

Making STAGS on a Linux system (as and if necessary)

For Linux installations, STAGS is typically distributed with source code, object libraries, and executables for specific machine architectures. Normally, it is not necessary for the installing administrator or for the typical user to “make” the STAGS programs. In unusual circumstances—such as when inconsistencies exist between the environment used to create the STAGS distribution software and that which is resident on the user’s machine—it may be necessary to do so. The computationally-sophisticated user may elect to recompile and re-link STAGS components to take advantage of high-performance compilation and/or optimization features that may be available on his or her machine.

To make STAGS, simply execute the make command from the $STAGSHOME directory. To make specific libraries or executables, issue the following command from any directory:
% makestags [target]

See the file $STAGSHOME/prc/makestags for a list of valid targets. To remake the star library, for example, execute the command:

% makestags star.a

The makestags command, with no arguments, will make all of STAGS. It is equivalent to executing the make command from $STAGSHOME.

Coping with installation errors (as and if necessary)

Some common causes of installation difficulties include insufficient disk space, lack of write permission in the selected directory, nonexistence of Linux commands that are required for the installation procedure, and installer boneheadedness. If installation is not successful, identify and correct the problem; then delete the entire $STAGSHOME directory and start over again at the 2.3.2 “Installation of STAGS on a Linux System” point.

Verifying that STAGS operates correctly

Correct operation of STAGS can be verified by running the small suite of test cases in the $STAGSHOME/testcases directory and examining selected portions of the output. Basic concepts and procedures for executing STAGS processors on Linux systems are described in the following subsection—which the installing administrator and anyone who uses STAGS on such a system must master. A first-level operation verification procedure for STAGS is discussed in Section 2.6 (at the end of this chapter).

2.3.3 Execution of STAGS on a Linux System

There are two basic types of executable STAGS programs:

- **default** executables
  these are the standard versions of the STAGS processors (s1, s2, stapl, scopy and xytrans) that are supplied with the program and used for most applications; these executables are always stored in the $STAGSHOME/bin directory

- **custom** executables
  these are application-dependent versions of s1, s2, stapl, scopy and/or xytrans—which are typically created by linking with user-written subroutines and/or user-defined elements and are typically stored in user-controlled directories.

On Linux systems, the $STAGSHOME/prc directory contains a number of procedure files that facilitate the execution and construction of STAGS programs. These procedure files (scripts) are described in the $STAGSHOME/README file. The most important of these (in the “execution” department) is the stags procedure—which is typically used to run
to construct a STAGS model, to perform one or more analyses with it, to perform pre- and/or post-analysis plotting operations, and to do other things with data on STAGS I/O files.* The most important of these procedures (in the “construction” department) is makeuser—which is used to “make” custom versions of STAGS processors when it is necessary to do so. The remainder of this subsection is devoted to descriptions of these two procedures:

**stags:** On Linux systems, one typically “runs” the STAGS program(s) by invoking the $STAGSHOME/prc/stags procedure; this is typically done via the stags alias, which $STAGSHOME/prc/initialize constructs when the user initializes STAGS prior to exercising it; the stags procedure can be used to run STAGS with default and/or with custom STAGS executables.

See “Executing STAGS with stags (Linux)” on page 2-22 for information about the stags procedure, and “Initializing STAGS on a Linux system” on page 2-19 for more information about the initialization process.

**makeuser** Custom, user-owned versions of executable STAGS processors may be created with the $STAGSHOME/prc/makeuser procedure, which is generally executed via the makeuser alias that $STAGSHOME/prc/initialize defines when the user initializes STAGS prior to exercising it; custom executables must be created and executed when user-written subroutines and/or user-defined elements are employed.

See “Creating custom STAGS executables with makeuser (Linux)” on page 2-23 for more information about the makeuser procedure.

See “Initializing STAGS on a Linux system” on page 2-19 for information about the initialization process.

See Chapter 12 “User-Written Subroutines” for more information about user-written subroutines, and Chapter 13 “User–Defined Elements” for more information about user-defined elements.

Users can also create special-purpose versions of STAGS code, since source code is distributed with the program; this type of advanced use is outside the scope of this document, but the sophisticated analyst may find occasion to customize STAGS for specific applications.

* See Chapter 17 “Input/Output Files” for documentation on STAGS I/O files.
Executing STAGS with stags (Linux)

Synopsis

```
stags [options] casename [options]
```

Options

- `-b` — run STAGS in background; the default nice value is 10.
- `-n [nice_value]` — set nice value for non-queued jobs; if option `-n` is selected, but a nice value is not supplied, the default is 10.
- `-sm` — send email message upon termination of background jobs; default is send no email: queued jobs always send email upon termination.
- `-1 [s1_executable]` — run s1; a user-specified s1 executable is optionally defined.
- `-2 [s2_executable]` — run s2; a user-specified s2 executable is optionally defined.

Examples

- run the default s1 and s2 executables interactively or in the foreground
  ```
  % stags casename
  ```
  If neither `-1` nor `-2` is present, then both s1 and s2 will be run in success, using the default executables.

- run a custom version of s2 with the default version of s1
  ```
  % stags -1 -2 [s2_executable] casename
  ```

- run a custom version of s1 with the default version of s2
  ```
  % stags -2 -1 [s1_executable] casename
  ```

- run default s1 and custom s2 in background, then send email upon completion
  ```
  % stags -b -1 -2 ~/my_s2 -sm mycase
  ```

- run custom s2 in the background, then send mail upon completion
  ```
  % stags -b -sm -2 ~/my_s2 mycase
  ```

- run s1 in the foreground
  ```
  % stags mycase -1
  ```

Note that options `-1`/`-2` appear after `mycase` in the above/below examples; any argument following `-1` or `-2` must be either a valid STAGS executable (custom or default) or another option (i.e., an argument beginning with a `-`).

- run s2 in the foreground
  ```
  % stags mycase -2
  ```
Creating custom STAGS executables with makeuser (Linux)

Synopsis

```
makeuser [-g] [-l userlib.a] [target]
```

Options

- **-g**  
  compile .F files with standard debug option

- **-cpp “cppflags”**  
  where cppflags are valid cpp options; user-entered cppflags are both prepended and appended to the default cppflags; the user-entered cppflags are prepended so that, if given, alternate include-file directories can be searched before the standard include-file directories; the user-entered cppflags are also appended so that default cpp definitions can be reset by the user; the user-entered cppflags do not replace the default cppflags because STAGS code will not compile correctly without certain default cpp definitions that the user is likely to forget to include.

  Example:  `-cpp "-D_debug_"`

- **-fc “fcflags”**  
  where user-entered fcflags (FORTRAN compiler options) replace the default compiler options for all modes of compilation: no optimization, partial optimization, full optimization, and debug; on most machines, users should remember to include the “-c” flag if they exercise this option.

  Example:  `-fc "-c -g -O2"`

- **-cc “ccflags”**  
  where user-entered ccflags (C compiler options) replace the default compiler options.

  Example:  `-cc "-c -g -O2"`

- **-ld “linkflags”**  
  where user-entered linkflags (linker options) replace the default linker options.

  Example:  `-ld "-M -Bstatic"

- **-l userlib.a**  
  Link library userlib.a when target is made; more than one library can be linked using additional “-l” arguments.

**target** is a target in $STAGSHOME/make/makefile.user. Valid targets are:

- **s1**  
  creates a custom version of **s1**, the STAGS model processor

- **s2**  
  creates a custom version of **s2**, the STAGS solution processor

- **scopy**  
  creates a custom version of **scopy**, a STAGS data postprocessor

- **pitrans**  
  creates a custom version of **pitrans**, a STAGS translator/postprocessor

- **xytrans**  
  creates a custom version of **xytrans**, a STAGS xy-plot-data translator

- **clean**  
  removes custom object and executable files from the current working directory

If no target is specified, then makeuser attempts to make all STAGS executables, placing them in the current working directory (see Table 2.2).

Each user-written subroutine must be in a .F source file. Routines called by user-written subroutines, if any, can be in .c files, .F files, and/or in .a library files; do not duplicate any STAGS names.
The `makeuser` procedure compiles source files (.F, .c, and .h, as appropriate) in the current working directory and links these files ahead of the standard STAGS libraries. If user libraries are included, they will be linked after any source files in the current directory but before any STAGS libraries.

The `makeuser` procedure operates exclusively on files that are in the current working directory ($cwd) and has no effect on the `$STAGSHOME` file system or on any other file system.

**CAUTION:** Existing .o files are erased by `makeuser`. These may be placed in a .a library file, which may be linked with the “-l” option.

`makeuser` attempts to “force link” any object modules produced from source files, and some linkers will abort if asked to force link a name that is not referenced. This vexation can be
avoided by ensuring that $cwd$ includes only those source files containing program modules whose names are referenced by the target.

**Examples**

Suppose the current working directory contains the following files:

```
-rw-r--r--  1 stags       12833 Jan 12 10:30 mylib1.a
-rw-r--r--  1 stags       78294 Jan 12 10:30 mylib2.a
-rw-r--r--  1 stags        1242 Jan 12 10:30 user2.F
-rw-r--r--  1 stags        7231 Jan 12 10:30 user1.F
```

- create a custom version of $s2$, incorporating user1.F and user2.F
  ```
  % makeuser s2
  ```
- create a custom version of $s2$—with user1.F and user2.F compiled for symbolic debugging
  ```
  % makeuser -g s2
  ```
- create a custom version of $s2$—linking in mylib1.a and mylib2.a, in addition to user1.F and user2.F
  ```
  % makeuser -l mylib1.a -l mylib2.a s2
  ```
- create a custom version of $s2$—linking in mylib1.a and mylib2.a, together with user1.F and user2.F compiled for symbolic debugging;
  ```
  % makeuser -g -l mylib1.a -l mylib2.a s2
  ```

In all of these examples, `makeuser` will produce a user.a user–library file and a `us2` user–executable file. Since no `.c` files and no `.h` files exist in the current directory, `makeuser` will also create dummy files dummyc.c and dummyi.h.
2.4 Macintosh-Based Systems

PRELIMINARY:

The good news here is that STAGS can be installed and executed on Macintosh-based machines. The bad news here is that this seriously underfunded section of Chapter 2 is also “under construction” and is also incomplete. We recommend that anyone who experiences difficulties in installing and executing STAGS on a Macintosh system contact Dr. Charles C. Rankin, at

Rhombus Consultants Group
2565 Leghorn Street
Mountain View, CA 94043

2.4.1 Basic Macintosh System Requirements

Basic system requirements that must be met to install STAGS on a Macintosh system are summarized in the following:

- Operating system: OS X (10.2)
- Minimum disk space required: 85 MB
- System utilities required: fpp & fsplit (supplied)
- Compilers required: gcc & g77

System requirements that must be met (and should be comfortably exceeded) to execute STAGS programs on a Macintosh system are summarized in the following table:

- Processor speed requirements: 300 MHz
- Memory requirements: 256 MB
- Storage requirements: 512 MB

2.4.2 Installation of STAGS on a Macintosh system

To install STAGS on a Macintosh system, one must be in a terminal window.

Installing STAGS on a Macintosh system

Macintosh versions of STAGS are typically distributed on CDs or via “ftp” as a pair of gzip-compressed archive files—the primary distribution file (stags_mac.tgz) and a secondary (but necessary) utility file (ftp_split.tgz). Installation of STAGS on a Macintosh system is typically a six-step process:
1) Establish the root directory into which STAGS is to be installed (the /home/stags directory, for example)—creating it as and if necessary; make this directory the current working directory.

2) Copy the two STAGS distribution files from their distribution medium into this working directory.

3) Extract STAGS from the stags_mac.tgz file and the fpp and fsplit utilities from the fpp_fsplit.tgz file by issuing the following two commands:

   % tar xzf stags_mac.tgz
   % tar xzf fpp_fsplit.tgz

   The first operation in this step creates a new subdirectory (named stags) within the current working directory. This new subdirectory is called $STAGSHOME in the directory structure diagram that is shown in Figure 2.1. After creating this new subdirectory, the first operation constructs the STAGS program system within it.

   The second operation in this step creates two additional subdirectories (named fpp and fsplit) within the current working directory. The fpp and fsplit subdirectories contain two utility programs that are needed whenever it is necessary to compile STAGS routines and/or to “make” STAGS executables.

4) Continue the installation process by executing the RUN.ME.FIRST script from the new stags subdirectory, as follows:

   % cd stags
   % RUN.ME.FIRST

   The RUN.ME.FIRST script modifies the STAGS initialization file (discussed next); then it commits suicide by deleting itself.

5) Continue the installation process by transferring from the stags subdirectory to fpp (one of stags’ two siblings) and performing the following operations:

   % cd ../fpp
   % make fpp

   The make fpp command compiles and “makes” the fpp utility program, producing fpp*. Transfer (copy or move) fpp* from the current working directory into the /usr/local/bin directory (which must be in the user’s path).

6) Conclude the installation process by transferring from the fpp subdirectory to fsplit (stags’ other sibling) and performing the following operations:

   % cd ../fsplit
   % make fsplit

   The make fsplit command compiles and “makes” the fsplit utility program, producing fsplit*. Transfer (copy or move) fsplit* from the current working directory into the /usr/local/bin directory (which must be in the user’s path).
Installation is now complete. An initialization procedure must be performed, however, before STAGS can be executed.

**Initializing STAGS on a Macintosh system**

Issue the following command once per login session prior to changing or running STAGS:

```
% source $STAGSHOME/prc/initialize
```

where $STAGSHOME is the complete path name to the stags directory structure that is shown in Figure 2.1. The $STAGSHOME/prc/initialize procedure establishes a number of command aliases and environmental variables that are necessary to operate on and/or to execute the STAGS programs. For example, $STAGSHOME is defined as an environmental variable that contains the path name of the stags directory.

It is strongly recommended that each STAGS user inspect the $STAGSHOME/prc/initialize file to see the aliases and variables that it creates, and install this command (or define an alias for it) in his or her .login file.

**Making STAGS on a Macintosh system (as and if necessary)**

For Macintosh installations, STAGS is typically distributed with source code and with object libraries and executables, for specific machine architectures. Normally, it is not necessary for the installing administrator or for the typical user to “make” the STAGS programs. In unusual circumstances—such as when inconsistencies exist between the environment used to create the STAGS distribution software and that which is resident on the user’s machine—it may be necessary to do so. The computationally-sophisticated user may elect to recompile and re-link STAGS components to take advantage of high-performance compilation and/or optimization features that may be available on his or her machine.

To make STAGS, simply execute the make command from the $STAGSHOME directory. To make specific libraries or executables, issue the following command from any directory:

```
% makestags [target]
```

See the file $STAGSHOME/prc/makestags for a list of valid targets. To remake the star library, for example, execute the command:

```
% makestags star.a
```

The makestags command, with no arguments, will make all of STAGS. It is equivalent to executing the make command from $STAGSHOME.
Coping with installation errors (as and if necessary)

Some common causes of installation difficulties include insufficient disk space, lack of write permission in the selected directory, nonexistence of Macintosh commands that are required for the installation procedure, and installer boneheadedness. If installation is not successful, identify and correct the problem; then delete the entire $STAGSHOME directory and start over again at the 2.4.2 “Installation of STAGS on a Macintosh system” point.

Verifying that STAGS operates correctly

Correct operation of STAGS can be verified by running the small suite of test cases in the $STAGSHOME/testcases directory and examining selected portions of the output. Basic concepts and procedures for executing STAGS processors on Macintosh systems are described in the following subsection—which the installing administrator and anyone who uses STAGS on such a system must master. A first-level operation verification procedure for STAGS is discussed in Section 2.6 (at the end of this chapter).

2.4.3 Execution of STAGS on a Macintosh System

There are two basic types of executable STAGS programs:

- **default executables** these are the standard versions of the STAGS processors (s1, s2, stapl, scopy and xytrans) that are supplied with the program and used for most applications; these executables are always stored in the $STAGSHOME/bin directory

- **custom executables** these are application-dependent versions of s1, s2, stapl, scopy and/or xytrans—which are typically created by linking with user-written subroutines and/or user-defined elements and are typically stored in user-controlled directories.

On Macintosh systems, the $STAGSHOME/prc directory contains a number of procedure files that facilitate the execution and construction of STAGS programs. These procedure files (scripts) are described in the $STAGSHOME/README file. The most important of these (in the “execution” department) is the stags procedure—which is typically used to run
to construct a STAGS model, to perform one or more analyses with it, to perform pre- and/or post-analysis plotting operations, and to do other things with data on STAGS I/O files.* The most important of these procedures (in the “construction” department) is makeuser—which is used to “make” custom versions of STAGS processors when it is necessary to do so. The remainder of this subsection is devoted to descriptions of these two procedures:

**stags:** On Macintosh systems, one typically “runs” the STAGS program(s) by invoking the $STAGSHOME/prc/stags procedure; this is typically done via the stags alias, which is constructed by $STAGSHOME/prc/initialize when the user initializes STAGS prior to exercising it; the stags procedure can be used to run STAGS with default and/or with custom STAGS executables.

See “Executing STAGS with stags (Macintosh)” on page 2-31 for information about the stags procedure, and “Initializing STAGS on a Macintosh system” on page 2-28 for more information about the initialization process.

**makeuser** Custom, user-owned versions of executable STAGS processors may be created with the $STAGSHOME/prc/makeuser procedure, which is generally executed via the makeuser alias that $STAGSHOME/prc/initialize defines when the user initializes STAGS prior to exercising it; custom executables must be created and executed when user-written subroutines and/or user-defined elements are employed.

See “Creating custom STAGS executables with makeuser (Macintosh)” on page 2-32 for more information about the makeuser procedure.

See “Initializing STAGS on a Macintosh system” on page 2-28 for information about the initialization process.

See Chapter 12 “User-Written Subroutines” for more information about user-written subroutines, and Chapter 13 “User–Defined Elements” for more information about user-defined elements.

Users can also create special-purpose versions of STAGS code, since source code is distributed with the program; this type of advanced use is outside the scope of this document, but the sophisticated analyst may find occasion to customize STAGS for specific applications.

* See Chapter 17 “Input/Output Files” for documentation on STAGS I/O files.
Executing STAGS with stags (Macintosh)

Synopsis

```
stags [options] casename [options]
```

Options

- `-b` — run STAGS in background; the default nice value is 10.
- `-n [nice_value]` — set nice value for non-queued jobs; if option `-n` is selected, but a nice value is not supplied, the default is 10.
- `-sm` — send email message upon termination of background jobs; default is send no email: queued jobs always send email upon termination.
- `-1 [s1_executable]` — run `s1`; a user-specified `s1` executable is optionally defined.
- `-2 [s2_executable]` — run `s2`; a user-specified `s2` executable is optionally defined.

Examples

- run the default `s1` and `s2` executables interactively or in the foreground
  
  ```
  % stags casename
  ```

  If neither `-1` nor `-2` is present, then both `s1` and `s2` will be run in succession, using the default executables.

- run a custom version of `s2` with the default version of `s1`
  
  ```
  % stags -1 -2 [s2_executable] casename
  ```

- run a custom version of `s1` with the default version of `s2`
  
  ```
  % stags -2 -1 [s1_executable] casename
  ```

- run default `s1` and custom `s2` in background, then send email upon completion
  
  ```
  % stags -b -1 -2 ~/my_s2 -sm mycase
  ```

- run custom `s2` in the background, then send mail upon completion
  
  ```
  % stags -b -sm -2 ~/my_s2 mycase
  ```

- run `s1` in the foreground
  
  ```
  % stags mycase -1
  ```

  Note that options `-1/-2` appear after `mycase` in the above/below examples; any argument following `-1` or `-2` must be either a valid STAGS executable (custom or default) or another option (i.e., an argument beginning with a `-`).

- run `s2` in the foreground
  
  ```
  % stags mycase -2
  ```
Creating custom STAGS executables with makeuser (Macintosh)

Synopsis

```
makeuser [-g] [-l userlib.a] [target]
```

Options

- `g` = compile .F files with standard debug option
- `cpp "cppflags"` = where cppflags are valid cpp options; user-entered cppflags are both **prepended** and **appended** to the default cppflags; the user-entered cppflags are prepended so that, if given, alternate include-file directories can be searched before the standard include-file directories; the user-entered cppflags are also appended so that default cpp definitions can be reset by the user; the user-entered cppflags do not replace the default cppflags because STAGS code will not compile correctly without certain default cpp definitions that the user is likely to forget to include.
  
  Example: `cpp "-D_debug_"`

- `fc "fcflags"` = where user-entered fcflags (FORTRAN compiler options) replace the default compiler options for all modes of compilation: no optimization, partial optimization, full optimization, and debug; on most machines, users should remember to include the “-c” flag if they exercise this option.
  
  Example: `fc "-c -g -O2"

- `cc "ccflags"` = where user-entered ccflags (C compiler options) replace the default compiler options.
  
  Example: `cc "-c -g -O2"

- `ld "linkflags"` = where user-entered linkflags (linker options) replace the default linker options.
  
  Example: `ld "-M -Bstatic"

- `-l userlib.a` = Link library userlib.a when target is made; more than one library can be linked using additional “-l” arguments.

`target` is a target in `$STAGSHOME/make/makefile.user`. Valid targets are:

- `s1` creates a custom version of `s1`, the STAGS model processor
- `s2` creates a custom version of `s2`, the STAGS solution processor
- `scopy` creates a custom version of `scopy`, a STAGS data postprocessor
- `pitrans` creates a custom version of `pitrans`, a STAGS translator/postprocessor
- `xytrans` creates a custom version of `xytrans`, a STAGS xy-plot-data translator
- `clean` removes custom object and executable files from the current working directory

If no `target` is specified, then `makeuser` attempts to make all STAGS executables, placing them in the current working directory (see Table 2.3).
Each user-written subroutine must be in a .F source file. Routines called by user-written subroutines, if any, can be in .c files, .F files, and/or in .a library files; do not duplicate any STAGS names.

The `makeuser` procedure compiles source files (.F, .c, and .h, as appropriate) in the current working directory and links these files ahead of the standard STAGS libraries. If user libraries are included, they will be linked after any source files in the current directory but before any STAGS libraries.

The `makeuser` procedure operates exclusively on files that are in the current working directory ($cwd) and has no effect on the $STAGSHOME file system or on any other file system.
CAUTION: Existing .o files are erased by `makeuser`. These may be placed in a .a library file, which may be linked with the “-l” option.

`makeuser` attempts to “force link” any object modules produced from source files, and some linkers will abort if asked to force link a name that is not referenced. This vexation can be avoided by ensuring that `$cwd` includes only those source files containing program modules whose names are referenced by the `target`.

**Examples**

Suppose the current working directory contains the following files:

- `-rw-r--r-- 1 stags       12833 Jan 12 10:30 mylib1.a`
- `-rw-r--r-- 1 stags       78294 Jan 12 10:30 mylib2.a`
- `-rw-r--r-- 1 stags        1242 Jan 12 10:30 user2.F`
- `-rw-r--r-- 1 stags        7231 Jan 12 10:30 user1.F`

- create a custom version of `s2`, incorporating `user1.F` and `user2.F`
  
  `% makeuser s2`

- create a custom version of `s2`—with `user1.F` and `user2.F` compiled for symbolic debugging
  
  `% makeuser -g s2`

- create a custom version of `s2`—linking in `mylib1.a` and `mylib2.a`, in addition to `user1.F` and `user2.F`
  
  `% makeuser -l mylib1.a -l mylib2.a s2`

- create a custom version of `s2`—linking in `mylib1.a` and `mylib2.a`, together with `user1.F` and `user2.F` compiled for symbolic debugging;
  
  `% makeuser -g -l mylib1.a -l mylib2.a s2`

In all of these examples, `makeuser` will produce a `user.a` user–library file and a `us2` user–executable file. Since no `.c` files and no `.h` files exist in the current directory, `makeuser` will also create dummy files `dummyc.c` and `dummyi.h`. 
2.5 Windows-Based Systems

The current version of STAGS can be installed and executed on Windows-based machines, but we strongly recommend that anyone who wants to do that should seriously consider Linux-based alternatives. This section of Chapter 2 is not “under construction” yet, so we recommend that anyone who is determined to install and execute STAGS on a Windows system should contact Dr. Frank C. Weiler, at

Lockheed-Martin Missiles & Space Co., Inc.
ATC Org. L9-21; Bldg. 204
3251 Hanover Street
Palo Alto, CA 94304

2.5.1 Basic Windows System Requirements

2.5.2 Installation of STAGS on a Windows System

Installing STAGS on a Windows system

Initializing STAGS on a Windows system

Making STAGS on a Windows system (as and if necessary)

Coping with installation errors (as and if necessary)

Verifying that STAGS operates correctly

2.5.3 Execution of STAGS on a Windows System
2.6 Installation Verification

Correct operation of STAGS can be verified by running the small suite of test cases in the $STAGSHOME/testcases directory and examining portions of the output. The precise way in which this is done depends on the user’s platform (and is described in the appropriate section of this chapter, above. For simplicity, the first-level verification process for a UNIX-based system is described here. Administrators and users with other platforms can make the necessary adjustments to this narrative for the platform of interest.

Change your current working directory to $STAGSHOME/testcases and follow the instructions below. If all three test cases produce correct results, then you may assume that STAGS is operating correctly on your machine.

Test 1: sxybuck

This is a small test case in which the shear buckling setup, boundary conditions and behavior are examined for a model with exotic material layup angles.

Execute STAGS [see “Executing STAGS with stags (UNIX)” on page 2-12 (for example) for information about the stags command):

```
% stags sxybuck
```

Examine the eigenvalues in the sxybuck.out2 output file and compare them to the correct values shown below. The results should be accurate to 4 or 5 significant digits.

<table>
<thead>
<tr>
<th>CRITICAL LOAD FACTOR COMBINATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO.</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

Test 2: pcats

This standard plasticity test case, with a simple model, is frequently used to test the effects of variations in material and other parameters.

Execute STAGS:

```
% stags pcats
```

Execute xytrans:

```
% xytrans pcats <<EOF
3
f
q
q
EOF
```
The file *pcats.step* should look like the file below. Results should be accurate to 4 or 5 significant digits.

There are 13 loadsteps for this case.

<table>
<thead>
<tr>
<th>Loadstep</th>
<th>Load Factor A</th>
<th>Load Factor B</th>
<th>time</th>
<th># of modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.100000E-01</td>
<td>0.000000E+00</td>
<td>0.000000E+00</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.100000E-01</td>
<td>0.000000E+00</td>
<td>0.000000E+00</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.200000E-01</td>
<td>0.000000E+00</td>
<td>0.000000E+00</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0.312503E-01</td>
<td>0.000000E+00</td>
<td>0.000000E+00</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0.447805E-01</td>
<td>0.000000E+00</td>
<td>0.000000E+00</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0.607618E-01</td>
<td>0.000000E+00</td>
<td>0.000000E+00</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0.791830E-01</td>
<td>0.000000E+00</td>
<td>0.000000E+00</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0.986861E-01</td>
<td>0.000000E+00</td>
<td>0.000000E+00</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0.116296E+00</td>
<td>0.000000E+00</td>
<td>0.000000E+00</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0.123024E+00</td>
<td>0.000000E+00</td>
<td>0.000000E+00</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0.125273E+00</td>
<td>0.000000E+00</td>
<td>0.000000E+00</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0.127322E+00</td>
<td>0.000000E+00</td>
<td>0.000000E+00</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0.128000E+00</td>
<td>0.000000E+00</td>
<td>0.000000E+00</td>
<td>0</td>
</tr>
</tbody>
</table>

**Test 3: dynela**

This is a small test case for an elastica model experiencing nonlinear dynamic transient response.

Execute STAGS:

```
% stags dynela
```

Execute xytrans:

```
% xytrans dynela <<EOF
4
f
11
q
q
EOF
```

Issue the UNIX command:

```
% head -4 dynela.velo.11
```

The lines below should be printed to your screen. Again, results should be accurate to 4 or 5 significant digits.
Getting Started

The first part of this chapter (Sections 3.1 through 3.6) presents a quick overview of STAGS by guiding the user step-by-step through an example nonlinear analysis. Though everything here is covered in more detail in other parts of the manual, this material provides a convenient reference to facilitate comprehension of the detailed explanations given in the chapters that follow. While it will be most useful to the new user, even an experienced user will benefit from a study of this example. The final part of this chapter (Section 3.7) is a brief description of some of the test problems that STAGS developers employ to verify that the program is working correctly and that STAGS users should study to become more familiar and confident with the program.

Some key concepts which are introduced in the following example problem are summarized below. These concepts should be kept in mind as the example problem is followed. At this point, details are unimportant, and the reader should not be concerned with understanding every aspect of the example analysis. What is important is to get a feel for what STAGS does and how it interacts with the user. It will be helpful to refer back to this chapter, scrutinizing it thoroughly, as other parts of the manual are absorbed.

- model definition
  Models are defined in a model input file, casename.inp, where casename is a character identifier supplied by the user. The model input file is referred to as the INP file. See Chapters 5–10.

- solution specification
  The analysis type and solution strategy are specified in a solution input file, casename.bin. The solution input file is referred to as the BIN file. Just as there are separate input files for model definition and solution specification, there are separate executable programs (s1 & s2) for the two analysis phases. The concept of independent model definition and solution specification is fundamental to the
tremendous power and flexibility provided by STAGS to restart an interrupted analysis, even switching analysis types (as from transient to static) at arbitrary points in the solution process. See Chapter 11 “Solution Input”.

- file systems

STAGS uses various file systems: text input/output files, binary database files, and scratch files. Two text input files (INP & BIN) are mentioned above. Understanding these file systems is important for effective use of STAGS. This is especially so for restarting an analysis, for pre/postprocessing, and for interfacing with the database via STAR, the STAGS Access Routines. See Chapter 17 “Input/Output Files”.

- program execution

STAGS execution involves various command procedures which manipulate I/O files and run the STAGS executables: s1, the model processor, and s2, the solution processor. When using user-written subroutines, custom, application-dependent executables are run in lieu of the standard ones. See Chapter 2 “Installation and Execution”.

### 3.1 Infinite Cylinder Under Hydrostatic Pressure

The infinite cylinder is a structure that is of great theoretical and practical interest. Figure 3.1 shows a finite-element model of a short segment from the middle of a cylinder of infinite length. It is a right circular cylinder, with a radius of \( R = 10.0 \text{ inches} \) and a wall thickness of \( t = 0.1 \text{ inch} \). The resulting ratio \( R/t = 100 \) indicates a thin-shell approximation. “Symmetry” boundary conditions (BCs) on the \( \theta = 0^\circ \) & \( \theta = 90^\circ \) planes permit a 360° structure to be modeled with a 90° arc, and the symmetry BCs on the \( x = 0 \) & \( x = L \) planes permit a cylinder of infinite length with a symmetric response to be modeled as one having a finite length. The applied loading is a ‘live’ hydrostatic pressure of \( p = 1.0 \text{ psi} \), acting in the inward radial direction. Live pressure is a STAGS traction loading which remains normal to the deformed surface throughout geometrically-nonlinear deformations. This “hydrostatic” pressure ignores the \( N_a = pR/2 \) end load since axial motion is constrained at both ends. The solution provided by thin-shell theory is

\[
\Delta R = (1 - \nu^2)pR^2/Et \\
\epsilon_c = \Delta R/R \\
\sigma_c = pR/t \\
\epsilon_a = 0 \\
\sigma_a = \nu\sigma_c
\]

where the subscript \( c \) indicates the circumferential direction, and \( a \) the axial direction.
A perfect cylinder (one whose shape is perfectly cylindrical with an absolutely uniform wall thickness) fabricated from an ideal material (one which remains linearly elastic under arbitrary strain states) will remain stable as it undergoes large displacements caused by increasing hydrostatic pressure. Eventually, the perfect cylinder will be compressed until its radius becomes zero, maintaining linear strain-displacement and constitutive behavior throughout. For such a structure, a linear static analysis will determine the load level corresponding to a given stress criteria, and a linear eigenanalysis will predict bifurcation-buckling load levels. Often, such linear analyses are adequate to determine allowable loads on a structure, but some applications warrant a more rigorous approach.

Real cylinders do not have perfect geometry and are not made of ideal materials. Moreover, some thin-shell structures buckle at fractions of their design loads, and their utility depends upon post-buckling strength—a linear analysis is of limited value for such a structure. Our analysis objective is to find the nonlinear collapse mode of such an imperfect cylinder. A rigorous collapse analysis must account for initial geometric imperfections as well as nonlinear behavior. The example presented here accounts for initial geometric imperfections and nonlinear strain-displacement relations, but does not include nonlinear material behavior. Applied to the analysis of thin shells, this strategy often reveals instability at load levels lower than those predicted by a linear buckling analysis, while stress levels are well below the yield criterion for the material.

**Figure 3.1** Example: Infinite cylinder under “live” hydrostatic pressure.
The solution strategy followed in our example problem consists of three phases:

- linear buckling analysis
  The first two eigenmodes are computed based upon a linear stress state. A linear combination of the eigenvectors will be used to specify an initial geometric imperfection for the nonlinear analysis, which follows.

- begin nonlinear static analysis
  The nonlinear analysis begins with a small load magnitude and proceeds up to a level just below the critical buckling load. Behavior is essentially linear in this region, but the effects of the geometric imperfections are important.

- restart nonlinear analysis
  The analysis restarts at the final load factor of the previous execution and continues through very large displacements as the cylinder collapses. The collapse mode is triggered by the presence of the initial geometric imperfections.

Each analysis phase is now described, in turn. Reference is made to various input files in the following descriptions. These input files are listed in Section 3.6 “Input Files” on page 3-10. Please refer to that section as needed throughout this chapter.

### 3.2 Linear Bifurcation Buckling

As explained in the introduction to this chapter, two text input files are required to perform an analysis. The INP file (casename.inp) describes the model, and the BIN file (casename.bin) specifies the solution strategy. The casename chosen for this analysis is “ring”. The INP file describing the model is given in “ring.inp.1” on page 3-11, and the BIN file is in “ring.bin.1”. Detailed explanations for the model and solution input are given in Chapter 5 “Model Input” through Chapter 11 “Solution Input”. It may be beneficial to preview these chapters now, to gain some understanding of the ring INP and BIN files. This is not essential, however.

The main point is that we are going to perform a buckling analysis of our cylindrical thin-shell structure. To do so, we must have an input file to describe the model and a second input file to specify the analysis type and solution strategy. So, let us perform the buckling analysis. First, create a scratch space to contain the files that will be used for our sample problem. For example, user “newton” might create the directory “/user/newton/stags/practice/ring”. Here the scratch directory will be referred to symbolically as “$RING”. Create a linear buckling analysis subdirectory, for example “$RING/buckling”, and in it install the input files for the buckling
analysis, copying them from the \texttt{\$STAGSHOME/examples/ring} directory, and giving them the required names of \texttt{casename.inp} and \texttt{casename.bin}. These operations are indicated symbolically using a metalanguage, as follows:

\begin{verbatim}
Copy \$STAGSHOME/examples/ring/ring.inp.1 to \$RING/buckling/ring.inp
Copy \$STAGSHOME/examples/ring/ring.bin.1 to \$RING/buckling/ring.bin
\end{verbatim}

At this point, the \$RING/buckling directory contains two files, \texttt{ring.inp} (the \textit{INP} file) and \texttt{ring.bin} (the \textit{BIN} file). Make \$RING/buckling the default directory, and execute the buckling analysis by issuing the \texttt{stags} command, giving “ring” (the \textit{casename}) as an argument. To run the buckling analysis interactively in the foreground, issue the following command:

\begin{verbatim}
stags ring
\end{verbatim}

For a complete discussion of the \texttt{stags} command, see \textquotedblleft Executing STAGS with \texttt{stags} (UNIX)	extquotedblright\ on page 2-12. On completion of the buckling analysis (which should take only a few seconds to perform), the \$RING/buckling directory will contain the following files:

- \texttt{ring.inp} model input
- \texttt{ring.bin} solution input
- \texttt{ring.m.pdf} model plot (PDF format)
- \texttt{ring.out1} model text output
- \texttt{ring.out2} solution text output
- \texttt{ring.sav} binary model data
- \texttt{ring.rst, ring.res, ring.imp, ring.egv} binary solution data
- \texttt{ring.log} execution summary text file

See \textquotedblleft File Systems\textquotedblright\ on page 3-8 and Chapter 17 \textquotedblleft Input/Output Files\textquotedblright\ for explanations of these and all \texttt{STAGS} files.

Plots of the first two buckling modes, which were just computed, are shown in Figure 3.2. For availability of graphical postprocessors for \texttt{STAGS}, see your site \texttt{STAGS} manager.

\* \texttt{\$STAGSHOME} is a system variable (a UNIX environmental variable) that refers to the top-level \texttt{STAGS} directory for your particular installation. See \textquotedblleft Initializing STAGS on a UNIX system\textquotedblright\ on page 2-8.
Next, we will begin the nonlinear collapse analysis, including initial geometric imperfections. It is necessary to create a new model in order to include the initial geometric imperfections. This new model is developed by modifying the previous (`ring.inp`) input file to account for the initial imperfections—simply by editing the existing file. Then, in order to run the nonlinear analysis, we must specify the analysis type and solution strategy. The `INP` file describing the model is given in “ring.inp.2” on page 3-12, and the `BIN` file is in “ring.bin.2”. Create a nonlinear analysis subdirectory, “$RING/collapse”, and in it install the input files for the nonlinear analysis, copying them from the `$STAGSHOME/examples/ring` directory, and giving them the required names of `casename.inp` and `casename.bin`. As before, this operation is indicated symbolically as:

\[
\begin{align*}
&\text{Copy } \$STAGSHOME/examples/ring/ring.inp.2 \text{ to } \$RING/collapse/ring.inp \\
&\text{Copy } \$STAGSHOME/examples/ring/ring.bin.2 \text{ to } \$RING/collapse/ring.bin \\
\end{align*}
\]

The initial geometric imperfections are defined as a linear combination of the first two buckling modes as

\[
\begin{align*}
\text{Mode 1, } \lambda = 2.76 & \\
\text{Mode 2, } \lambda = 13.8 & \\
\end{align*}
\]
\[ \Phi = \alpha_1 \phi_1 + \alpha_2 \phi_2 \]

where \( \phi_1 \) is the eigenvector corresponding to the first buckling mode, and \( \phi_2 \) is the second mode. The \( \alpha_i \) are constants, and \( \Phi \) is the resulting imperfection vector. Inspection of the INP file reveals that \( \alpha_1 = 0.0030 \) and \( \alpha_2 = 0.0015 \). With \( t = 0.10'' \), and with each eigenvector \( \phi_i \) scaled so that the largest translational component has a value of 1.0, it can be seen that the imperfection is specified to be a linear combination of the first buckling mode, having a maximum amplitude equal to 3.0% of the shell thickness, plus the second buckling mode, having a maximum amplitude equal to 1.5% of the shell thickness. All translational components of the eigenvector are used to generate the initial imperfection shape. This shape is then output in the model plot file (casename_m.pdf or casename_m.ps).

The eigenvectors are contained in the file pair (IMP, EGV) computed in the linear buckling analysis. These files must be present in the SRING/collapse directory since they are indirectly referenced in the INP file. Install them as indicated below:

```
COPY $RING/buckling/ring.imp TO $RING/collapse/ring.imp
COPY $RING/buckling/ring.egv TO $RING/collapse/ring.egv
```

To execute the first phase of the collapse analysis interactively, issue the following command in the $RING/collapse directory:

```
stags ring
```

Upon completion of this analysis (which should take only a few seconds to perform), the $RING/collapse directory will contain the following files:

- ring.inp: model input
- ring.bin: solution input
- ring_m.pdf: model plot (including imperfection)
- ring.out1: model text output
- ring.out2: solution text output
- ring.sav: binary model data
- ring.rst, ring.res, ring.imp, ring.egv: binary solution data
- ring.log: execution summary text file
3.4 Restarting a Nonlinear Analysis

A new BIN file is needed to restart the nonlinear collapse analysis. This is found in $STAGSHOME/examples/ring/ring.bin.3. Copy this file to the $RING/collapse directory, as follows:

Copy $STAGSHOME/examples/ring/ring.bin.3 to $RING/collapse/ring.bin

The above operation will overwrite the existing ring.bin, so save it with a different name, such as “ring.bin.first”, if desired. To resume the collapse analysis interactively, issue the following command in the $RING/collapse directory:

stags ring -2

After completion of the restarted analysis, (again, taking only a few seconds to perform), the $RING/collapse directory will contain the following files:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ring.inp</td>
<td>model input</td>
</tr>
<tr>
<td>ring.bin</td>
<td>solution input</td>
</tr>
<tr>
<td>ring_m.pdf</td>
<td>model plot (including imperfection)</td>
</tr>
<tr>
<td>ring.out1</td>
<td>model text output</td>
</tr>
<tr>
<td>ring.out2</td>
<td>solution text output</td>
</tr>
<tr>
<td>ring.out2.1</td>
<td>previous ring.out2 (renamed)</td>
</tr>
<tr>
<td>ring.sav</td>
<td>binary model data</td>
</tr>
<tr>
<td>ring.rst, ring.res, ring.imp, ring.egv</td>
<td>binary solution data</td>
</tr>
<tr>
<td>ring.log</td>
<td>execution summary text file</td>
</tr>
</tbody>
</table>

The final deformed shape is shown in Figure 3.3, and the load-deflection history is shown in Figure 3.4. Note that for a load factor of nearly four the radial displacement at $\theta = 0^\circ$ is about 3.7", and at $\theta = 90^\circ$, about 8.0". Compare these radial displacements with the cylinder radius, $R = 10"$ and shell thickness $t = 0.1"$.

3.5 File Systems

Table 3.1 is a summary of the files which are discussed in the foregoing example problem. This table is included as a quick reference to help introduce the user to STAGS file systems. For complete documentation of all STAGS files, see Chapter 17 “Input/Output Files”. The meaning of the information in the six-column format is as follows:
Figure 3.3  Deformed shape (true magnitude) at a load corresponding to $P_A = 4.0$. Compare with Figure 3.4, the load-displacement curve.

Figure 3.4  Load-displacement curves, showing radial displacement vs. load-factor $P_A$ at $\theta = 0^\circ$ & $\theta = 90^\circ$. 
• **file** *general* file name, used to refer to a file without reference to a specific *casename*.

• **name** *particular* file name, where *casename* is a user-assigned alphanumeric string of \( \leq 10 \) characters. *casename* is “ring” for the example problem.

• **type** either T for text (ASCII) or B for binary.

• **s1, s2** indicates action taken by each of the two processors: s1, the model processor, and s2, the solution processor. This will be one of four categories: R (read), W (write), R/W (read & write), or null (no action).

• **description** a brief explanation of the file contents

### Table 3.1 STAGS files.

<table>
<thead>
<tr>
<th>file</th>
<th>name</th>
<th>type</th>
<th>s1</th>
<th>s2</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INP</td>
<td>casename.inp</td>
<td>text</td>
<td>R</td>
<td></td>
<td>model input</td>
</tr>
<tr>
<td>BIN</td>
<td>casename.bin</td>
<td>text</td>
<td></td>
<td>R</td>
<td>solution input</td>
</tr>
<tr>
<td>OUT1</td>
<td>casename.out1</td>
<td>text</td>
<td>W</td>
<td></td>
<td>model output</td>
</tr>
<tr>
<td>OUT2</td>
<td>casename.out2</td>
<td>text</td>
<td></td>
<td>W</td>
<td>solution output</td>
</tr>
<tr>
<td>OUT2.i</td>
<td>casename.out2.i</td>
<td>text</td>
<td></td>
<td></td>
<td><em>OUT2</em>, previous run</td>
</tr>
<tr>
<td>LOG</td>
<td>casename.log</td>
<td>text</td>
<td></td>
<td></td>
<td>execution summary</td>
</tr>
<tr>
<td>SAV</td>
<td>casename.sav</td>
<td>binary</td>
<td>W</td>
<td>R</td>
<td>model data</td>
</tr>
<tr>
<td>PDF</td>
<td>casename_m.pdf</td>
<td>text</td>
<td>W</td>
<td></td>
<td>configuration plot</td>
</tr>
<tr>
<td>PS</td>
<td>casename_m.ps</td>
<td>text</td>
<td>W</td>
<td></td>
<td>configuration plot</td>
</tr>
<tr>
<td>RES</td>
<td>casename.res</td>
<td>binary</td>
<td></td>
<td>R/W</td>
<td>solution data</td>
</tr>
<tr>
<td>RST</td>
<td>casename.rst</td>
<td>binary</td>
<td></td>
<td>R/W</td>
<td>solution data addresses</td>
</tr>
<tr>
<td>EGV</td>
<td>casename.egv</td>
<td>binary</td>
<td></td>
<td>R/W</td>
<td>solution data</td>
</tr>
<tr>
<td>IMP</td>
<td>casename.imp</td>
<td>binary</td>
<td></td>
<td>R/W</td>
<td>solution data addresses</td>
</tr>
</tbody>
</table>

### 3.6 Input Files

The input files used in the example problem described in this section are supplied with the STAGS program system, located in the `$STAGSHOME/examples/ring` directory, with names as
summarized below. This section concludes with a listing of each of these files.

<table>
<thead>
<tr>
<th>analysis</th>
<th>model</th>
<th>solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear buckling analysis</td>
<td>ring.inp.1</td>
<td>ring.bin.1</td>
</tr>
<tr>
<td>start nonlinear analysis</td>
<td>ring.inp.2</td>
<td>ring.bin.2</td>
</tr>
<tr>
<td>restart nonlinear analysis</td>
<td>—</td>
<td>ring.bin.3</td>
</tr>
</tbody>
</table>

**Linear buckling analysis**

**ring.inp.1**

Infinite Cylinder, Live Hydrostatic Pressure

```
0                   $ B-1
1                   $ B-2  1 Shell unit
1 0 1               $ B-3  1 material type, 1 wall type
2 10                $ F-1  2 rows, 10 columns for 10 degree hoop spacing
1                   $ I-1  Material number 1
1.E7 .3 0 .1        $ I-2  Modulus, Poisson’s ratio, density
1 1 1               $ K-1  Wall ID #, wall type, # layers
1 0.1               $ K-2  Layer 1 has material 1, thickness 0.1
$
$ Begin Shell Unit Definition
$
5                   $ M-1  Shell type 5 means CYLINDER
0. 5.   0. 90. 10. $ M-2A  Axial Coord. 1 and 2, 0. to 90. Deg
1                   $ M-5  Wall type 1 used
410                 $ N-1  The 410 element will be used
4 4 4 4             $ P-1  Symmetry on all boundaries
1                   $ Q-1  1 load system defined
1 1                 $ Q-2  1 record on A system
-1. 5 3 0 0         $ Q-3  Live hydrostatic pressure everywhere
0                   $ R-1  Print nothing except diagnostics
```

**ring.bin.1**

Linear Buckling Solution Control

```
1 1                   $ B-1  Do linear bifurcation and archive displs and modes
1.0                  $ C-1  Reference load set to unity
0                    $ D-2
2                    $ D-3  Two modes
```
Start nonlinear analysis

ring.inp.2

Infinite Cylinder, Small Imperfection, Live Hydrostatic Pressure
0 $ B-1
1 0 0 0 0 0 2 $ B-2 1 Shell unit and 2 imperfection modes read
1 0 1 $ B-3 1 material type, 1 wall type
.0030 0 1 1 $ B-5 Weight .0030, step 0, mode 1, run 1
.0015 0 2 1 $ B-5 Weight .0015, step 0, mode 2, run 1
2 10 $ F-1 2 rows, 10 columns for 10 degree hoop spacing
1 $ I-1 Material number 1
1. E7 .3 0. .1 $ I-2 Modulus, Poisson’s ratio, density
1 1 1 $ K-1 Wall ID #, wall type, # layers
1 0.1 $ K-2 Layer 1 has material 1, thickness 0.1
$
$ Begin Shell Unit Definition
$
$ 5 $ M-1 Shell type 5 means CYLINDER
0. 5. 0.90. 10. $ M-2A Axial Coord. 1 and 2, 0. to 90. Deg
1 $ M-5 Wall type 1 used
410 $ N-1 The 410 element will be used
4 4 4 4 $ P-1 Symmetry on all boundaries
1 $ Q-1 1 load system defined
1 1 $ Q-2 1 record on A system
-1. 5 3 0 0 $ Q-3 Live hydrostatic pressure everywhere
0 $ R-1 Print nothing except diagnostics

ring.bin.2

Nonlinear Static Analysis Solution Control
3 1 $ B-1 Nonlinear static analysis, saving displacements
0.5 0.5 2.7 $ C-1 Start load, delta-load, final load
0 300 12 -20 0 1.0E-6 $ D-1 Begin new case, 12 cuts, true Newton,
$ begin with load control, error tol. = 1.0E-6
0 $ ET-1 Default Riks arc-length control

Restart nonlinear analysis

ring.bin.3

Restart Nonlinear Static Analysis
3 1 $ B-1 restart nonlinear analysis, saving disp.
2.7 1.0 4.0 $ C-1 Start load, Riks scale factor, final load
15 300 12 -20 -1 1.0E-6 $ D-2 Restart at 15, 12 cuts, true Newton,
continue Riks, error tol. = 1.e-6
$
0 $ ET-1 Assume solution is continuous on restart
3.7 Test problems

As noted above, this section contains brief descriptions of some of the test problems that STAGS developers employ to verify that the program is working correctly and that STAGS users should study to become more familiar and confident with the program. The test cases subject is large and filled with complexities that cannot be discussed in any meaningful way within the severe constraints of this document—especially in view of the nonexistent funding for and the limited energies of individuals who might contribute to those efforts. The best that we can do under the circumstances at hand is to present the information in the following table, which contains one-line descriptions of (some of) the problems that STAGS developers exercise to determine whether or not the program is functioning properly after it has been modified and/or ported to another computer system. The input (and printed output) files for this suite of test cases can be found on the $STAGSHOME/testcases directory and on directories that are subordinate to it. We encourage the interested (or desperate) reader to study those cases that are of greatest interest to him/her and/or which have the most relevance to his/her problems.

Table 3.2 Suite of Test Cases for STAGS.

<table>
<thead>
<tr>
<th>case #</th>
<th>case name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>acoro</td>
<td>test plastic collapse capabilities (historical fabrication methods)</td>
</tr>
<tr>
<td>2</td>
<td>acoro_gcp</td>
<td>test plastic collapse capabilities (GCP fabrication methods)</td>
</tr>
<tr>
<td>3</td>
<td>annular</td>
<td>test pole functionality; test auto-generation of triangular elements at a pole</td>
</tr>
<tr>
<td>4</td>
<td>arch</td>
<td>test snap-through path-following under dead loading</td>
</tr>
<tr>
<td>5</td>
<td>arch1</td>
<td>test snap-through path-following under dead loading</td>
</tr>
<tr>
<td>6</td>
<td>bebox</td>
<td>test G-1 connections along various boundary lines (including ‘reverse’ numbering)</td>
</tr>
<tr>
<td>7</td>
<td>bplast</td>
<td>test beam plasticity; test plasticity tangent stiffness</td>
</tr>
<tr>
<td>8</td>
<td>bra_1</td>
<td>test planar boundary conditions; test solver treatment of Lagrange constraints; test path-following solution algorithm; test ‘cable moment’ loading</td>
</tr>
<tr>
<td>9</td>
<td>cone</td>
<td>test buckling eigensolver</td>
</tr>
<tr>
<td>10</td>
<td>contact1</td>
<td>test general point/surface contact algorithms</td>
</tr>
<tr>
<td>11</td>
<td>contact2</td>
<td>same as contact1, adding generalized fastener</td>
</tr>
<tr>
<td>12</td>
<td>contact3</td>
<td>test general point/surface contact algorithms for configuration with large contact region</td>
</tr>
<tr>
<td>case #</td>
<td>case name</td>
<td>description</td>
</tr>
<tr>
<td>--------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>13</td>
<td>disk</td>
<td>test response of disk to center load</td>
</tr>
<tr>
<td>14</td>
<td>disk1</td>
<td>same as disk, permitting only ( w ) at the center</td>
</tr>
<tr>
<td>15</td>
<td>doors</td>
<td>test G-2 partial compatibility connectivity specifications</td>
</tr>
<tr>
<td>16</td>
<td>dynela</td>
<td>test nonlinear dynamic transient response, using the Park-method integrator, for elastica problem</td>
</tr>
<tr>
<td>17</td>
<td>elplas</td>
<td>test multi-segment White-Besseling curve for plasticity problem</td>
</tr>
<tr>
<td>18</td>
<td>ep130</td>
<td>test plasticity option with E130 fastener elements</td>
</tr>
<tr>
<td>19</td>
<td>f15s</td>
<td>reference crack test: crack introduced by changing boundary conditions; test E510 mesh-transition elements in presence of stiffener blades; test continuities in stress plots (with stapl)</td>
</tr>
<tr>
<td>20</td>
<td>f15t</td>
<td>similar to f15s, with more complexities</td>
</tr>
<tr>
<td>21</td>
<td>f15t2</td>
<td>similar to f15s, without blade stiffeners</td>
</tr>
<tr>
<td>22</td>
<td>f15tr2</td>
<td>test mesh-transition boundaries with G-1 instead of G-2 compatibility specifications; test multiple transitions with re-entrant corner, modeled with E510 and E710 transition elements</td>
</tr>
<tr>
<td>23</td>
<td>fast2</td>
<td>simple fastener test case</td>
</tr>
<tr>
<td>24</td>
<td>fast3</td>
<td>test looping input in element unit with E130 (fastener) elements</td>
</tr>
<tr>
<td>25</td>
<td>fast4</td>
<td>test looping capabilities, with user points as well as elements</td>
</tr>
<tr>
<td>26</td>
<td>g2e410</td>
<td>test soccer-ball pinched-hemisphere problem; test warping</td>
</tr>
<tr>
<td>27</td>
<td>ground</td>
<td>test cylindrical panel with ground motions</td>
</tr>
<tr>
<td>28</td>
<td>he130</td>
<td>test E130 fastener elements with hyperelastic properties</td>
</tr>
<tr>
<td>29</td>
<td>indisp</td>
<td>test ring problem, with initial displacements</td>
</tr>
<tr>
<td>30</td>
<td>lamequad</td>
<td>test default Lame routine and G-1 connectivities, for triangle constructed entirely with quadrilateral plate elements</td>
</tr>
<tr>
<td>31</td>
<td>mountd</td>
<td>test use of mount tables and elements to reproduce inverse-square attraction forces; test stability of dynamic transient response algorithms under tough conditions</td>
</tr>
<tr>
<td>32</td>
<td>pad1</td>
<td>test E810 Pad Hertzian contact element with beam-hitting-beam problem</td>
</tr>
<tr>
<td>33</td>
<td>pad2</td>
<td>same as pad1, with a twist: test sideslip capability of E810 Pad element</td>
</tr>
<tr>
<td>34</td>
<td>pcats</td>
<td>standard plasticity test case; easy to alter to test effects of material and other parameters</td>
</tr>
</tbody>
</table>
The interested reader is also encouraged to spend time and energy studying the interesting and informative test cases that are documented (with varying levels of detail) in the *STAGS Test Cases Manual.*
4

Fundamentals

4.1 Coordinate Systems

Effective use of STAGS demands a thorough understanding of the many coordinate systems it employs to define shell structures. These systems are described in alphabetical order in this section. Table 4.1 on page 4-2 summarizes coordinate systems that STAGS uses, giving brief descriptions along with references to sketches where graphical representation of various coordinate systems can be found. A word of encouragement to the new user: do not be daunted by this list of coordinate systems. You will quickly discover that they are easy to understand and natural to use; and they are all orthogonal coordinate systems.

\[(x, y, z)\]  \textit{branch (or shell unit) coordinates}

Branch coordinate system, a Cartesian coordinate system that is defined independently for each shell unit. The geometry of the surface is determined by the relations

\[x = x(X, Y)\quad y = y(X, Y)\quad z = z(X, Y)\]

where \((X, Y)\) are surface coordinates. See “Standard Shell Surfaces” beginning on page 6-2, where these surface-to-branch relations are given for each of the standard shell types. Note that a shell unit is fixed with respect to its branch coordinate system.

\[(x'', y'', z'')\]  \textit{computational coordinates}

A rectilinear system which defines the directions of the displacement unknowns at all nodes in a unit. In shell units, including the user-generated shell unit \((\text{ISHELL} = 1, M-1)\), the \((x'', y'', z'')\) system is identical to the shell \((X', Y', Z')\) system. In element units, the auxiliary system \((x_a, y_a, z_a)\), where defined, determines the nodal freedoms; otherwise, the nodal-global system \((x_g, y_g, z_g)\) is used.
Table 4.1  STAGS coordinate systems.

<table>
<thead>
<tr>
<th>symbol</th>
<th>name</th>
<th>use</th>
<th>sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(x, y, z)$</td>
<td>branch</td>
<td>shell-unit definition</td>
<td>Figure 4.1 on page 4-6 “Standard Shell Surfaces” beginning on page 6-2</td>
</tr>
<tr>
<td>$(x'', y'', z'')$</td>
<td>computational</td>
<td>primary solution (nodal dof)</td>
<td>Figure 7.1 on page 7-6</td>
</tr>
<tr>
<td>$(\bar{y}, \bar{z})$</td>
<td>cross-section</td>
<td>beam cross-section</td>
<td>Figure 5.5 on page 5-122</td>
</tr>
<tr>
<td>$(x', y', z')$</td>
<td>element</td>
<td>element computations</td>
<td>Figure 8.1 on page 8-12</td>
</tr>
<tr>
<td>$(x_e, y_e, z_e)$</td>
<td>element-edge</td>
<td>line loads</td>
<td>Figure 4.2 on page 4-7</td>
</tr>
<tr>
<td>$(x_s, y_s, z_s)$</td>
<td>element-surface</td>
<td>surface traction</td>
<td>Figure 4.2 on page 4-7</td>
</tr>
<tr>
<td>$(\bar{x}, \bar{y})$</td>
<td>fabrication</td>
<td>wall fabrication</td>
<td>Figure 5.7 on page 5-133</td>
</tr>
<tr>
<td>$(x_g, y_g, z_g)$</td>
<td>global</td>
<td>reference for all coordinate systems</td>
<td>Figure 4.1 on page 4-6</td>
</tr>
<tr>
<td>$(\phi_1, \phi_2)$</td>
<td>material</td>
<td>material properties</td>
<td>Figure 5.7 on page 5-133</td>
</tr>
<tr>
<td>$(x_a, y_a, z_a)$</td>
<td>nodal-auxiliary</td>
<td>option for nodal dof, element unit</td>
<td>Figure 7.1 on page 7-6</td>
</tr>
<tr>
<td>$(\bar{x}_g, \bar{y}_g, \bar{z}_g)$</td>
<td>nodal-global</td>
<td>option for nodal dof, element unit</td>
<td>Figure 7.1 on page 7-6</td>
</tr>
<tr>
<td>$(X', Y', Z')$</td>
<td>shell</td>
<td>nodal dof, shell unit</td>
<td>Figure 4.1 on page 4-6</td>
</tr>
<tr>
<td>$(X, Y)$</td>
<td>surface</td>
<td>shell-unit surface parameters</td>
<td>Figure 4.1 on page 4-6</td>
</tr>
<tr>
<td>$(x_w, y_w)$</td>
<td>wall-reference</td>
<td>reference for fabrication system</td>
<td>Figure 6.2 on page 6-28</td>
</tr>
</tbody>
</table>
(\(\xi, \eta\))  \textit{cross-section coordinates}

A Cartesian system used in definition of beam cross-sections. For convenience, the beam axial direction is sometimes represented by the symbol \(\xi\), \((\xi, \eta, \zeta)\) forming a 3-D orthogonal system. For shell-unit stiffeners, \(\xi\) is parallel to \(\eta'\) for rings and to \(\eta'\) for stringers; see Figure 6.5 on page 6-52. For element-unit beams, \(\xi\) is parallel to the beam element \(x'\) axis; see Figure 8.1 on page 8-12.

\textit{NOTE:} Do not confuse \((\xi, \eta, \zeta)\) cross-section coordinates with \((\xi, \eta, \zeta)\) fabrication coordinates. The distinction will be clear from the context in which the particular coordinate system is referenced.

\((x', y', z')\)  \textit{element coordinates}

A Cartesian system defined independently for each element in both shell units and element units. Chapter 14 “The Element Library” contains an explanation for each element type. Generally, for shell elements, only the element normal \(z'\) is of importance to the user, since pressure is the only data for input or output that refers to these coordinates. Although various computations are performed in element coordinate systems, quantities are always transformed to other systems for presentation to the user. For nonlinear analysis using corotation, \((x', y', z')\) rotates rigidly with the element as it deforms.

\((x_e, y_e, z_e)\)  \textit{element-edge coordinates}

A local element system which is independently defined on each element edge. The \(y_e\) direction is defined by the vector pointing from edge-node 1 to edge-node 2, projected onto the element plane. \(z_e\) is the same as \(z'\), the element normal, and \(x_e\), directed outward from the edge, completes a right-handed system.

\((x_s, y_s, z_s)\)  \textit{element-surface coordinates}

A local element system in which the \(x_s\) direction is defined by the vector pointing from element-node 1 to element-node 2, projected onto the element plane. \(z_s\) is the same as \(z'\), the element normal; and \(y_s\), in the element plane, completes a right-handed system.

\((\xi, \eta)\)  \textit{fabrication coordinates}

A 2-D Cartesian system used as a reference for defining a wall fabrication. \((\xi, \eta)\) are in the plane of the shell or element, which is determined by the shell normal \(Z'\) in shell units and by the element normal \(z'\) in element units. For convenience, the normal direction is sometimes represented by the symbol \(\zeta\) (please see Figure 5.7 on
Fundamentals

Coordinate Systems

page 5-133), (x, y, z) forming a 3-D orthogonal system. The fabrication coordinate system is oriented in the surface-tangent plane by rotating the wall-reference system (x_w, y_w) through the angle ZETA, a right-handed rotation about z (see Figure 6.2 on page 6-28 and Figure 8.2 on page 8-19).

NOTE: Do not confuse (x, y, z) fabrication coordinates with (x̃, ỹ, z̃) cross-section coordinates. The distinction will be clear from the context in which the particular coordinate system is referenced.

(x̃_g, ỹ_g, z̃_g)  global coordinates
A global Cartesian coordinate system. The orientations of all local systems for shell units or the element units are defined with respect to this system.

(φ_1, φ_2)  material coordinates
A 2-D Cartesian system in the plane of the shell or element. The material properties of individual shell wall layers are defined in the (φ_1, φ_2) directions, which are established by rotating the fabrication coordinates (x̃, ỹ) through the angle ZETL, a right-handed rotation about ẑ, which is uniquely defined for each layer. See Figure 5.7 on page 5-133.

(x_a, y_a, z_a)  nodal-auxiliary coordinates
An arbitrary, rectangular system which may optionally be defined by the user at each node in an element unit. Where defined, the nodal-auxiliary system determines the directions of the computational freedoms (see the computational coordinates description, above), which are determined by the nodal-global system, defined next.

(x_g, y_g, z_g)  nodal-global coordinates
A Cartesian system with its origin at a finite element node and with coordinate axes parallel to (x̃_g, ỹ_g, z̃_g) (i.e., a simple translation of the global system to a particular node).

(X', Y', Z')  shell coordinates
A local rectilinear system defined at each point on the reference surface of shell units. The (X', Y') coordinates are tangents to the coordinate lines defined by the surface coordinates (X, Y). Z' is the shell normal, directed “outward”, outward being defined by the sense of the cross product (X' × Y'). Note that (X', Y', Z') have units of length, whereas (X, Y) are surface parameters whose units are dependent upon the specific shell type (see “Standard Shell
Surfaces” beginning on page 6-2). In a cylindrical shell, for example, $X$ is the meridional distance (axial length) and $Y$ is the circumferential distance (arc length in degrees).

For the standard STAGS shell units ($\text{ISHELL} = 2–12$; see “Standard Shell Surfaces” beginning on page 6-2), the $(X', Y', Z')$ shell coordinates naturally form an orthogonal set, with the single exception of the Elliptic Cone ($\text{ISHELL} = 9$). For the Elliptic Cone and for shell units that are generated by user-written subroutine LAME ($\text{ISHELL} = 1$), $(X', Y', Z')$ is orthogonalized in the following ways. $Z'$ is the surface normal in every case. For $\text{ISHELL} = 9$, the circumferential $(Y)$ and shell-normal directions are chosen as the $(Y', Z')$ directions, and $Y' \times Z' = X'$ establishes the orthogonal system. For $\text{ISHELL} = 1$ (LAME), the user optionally selects either $X'$ or $Y'$ by reference to the corresponding surface coordinate ($X$ or $Y$). The remaining axis is computed to establish a right-handed system.

**NOTE:** For convenience, the symbol $Z'$ sometimes is used to indicate “either the shell normal $Z'$ or the element normal $z'$, as applicable.” For example, see Figure 5.6 on page 5-130. The perceptive reader will have no difficulty making this distinction.

**surface coordinates**

Basic curvilinear surface coordinates for shell units. A pair of values $(X, Y)$ specifies the position of a point on the shell reference surface. These coordinates also define the directions of the in-plane displacements $(u, v)$, and may also be used for grid generation. These are parametric coordinates on a plane that is mapped onto a two dimensional surface imbedded in three dimensional space. An example is the mapping of the spherical angles $(\psi, \theta)$ into three-space in the generation of a spherical shell. The mapping of $(X, Y)$ into branch coordinates $(x, y, z)$ for all the standard STAGS shell units can be found in “Standard Shell Surfaces” beginning on page 6-2. $(X, Y)$ form an orthogonal set, with the single exception of the Elliptic Cone ($\text{ISHELL} = 9$).

**wall-reference coordinates**

A 2-D Cartesian system used as a reference for orienting a wall fabrication in the plane of the shell or element. By default, $(x_w, y_w)$ are coincident with the $(X', Y')$ shell coordinates in shell units (see Figure 6.2 on page 6-28) and are established by selecting and projecting one of the $(x_g, y_g)$ global direction vectors onto the element surface in element units (see Figure 8.2 on page 8-19). For both shell units and element units, there is an option for establishing the wall-reference system by projecting a user-defined
“wall-reference vector” onto the element surface. See Figure 8.3 on page 8-20 for element units; this option is currently unavailable for shell units.

Figure 4.1 Shell-unit coordinate systems.
Figure 4.2  Element coordinate systems.
4.2 **Shell Unit**

STAGS models are composed of one or more individual *substructures*. Each substructure is either a *shell unit* or an *element unit*. A shell unit (a quadratic surface which is automatically generated and meshed) is chosen from a library of “standard geometries”, while an element unit is an arbitrary assemblage of user-defined nodes and elements. There are no limitations placed on the number of substructures in a model.

The configuration of a shell unit is described by the use of a reference surface. The location of any point on the reference surface can be uniquely defined by use of a set of two independent parameters \((X, Y)\). The geometry of the reference surface is given after a functional relationship has been established that defines the coordinates in a Cartesian system \((x, y, z)\) as functions of the values of these two parameters:

\[
x = x(X, Y) \quad y = y(X, Y) \quad z = z(X, Y)
\]  \hspace{1cm} (4.1)

The parameters \((X, Y)\) defining the position of a point on a surface are referred to as *surface coordinates*, and \((x, y, z)\) are referred to as *branch coordinates*. For the shell and plate configurations contained in the library of standard geometries, the user need not be concerned with these relations.

If one of the surface coordinates \((X, Y)\) is held constant and the other is varied over a range, a trace is obtained on the surface. Such a trace is referred to as a coordinate line. Two sets of such coordinate lines are always used to define the surface grid needed in a finite element analysis.

For each shell unit, a unique branch coordinate system \((x, y, z)\) is defined to describe that unit, or *branch*. Another Cartesian system \((x_g, y_g, z_g)\) is selected as a *global coordinate* system. Special parameters are used to specify the orientation of each branch coordinate system with respect to the global system, unless the position of the unit is uniquely determined by connection to a previously defined shell unit.

At the intersection of grid lines on the shell reference surface (grid points), a local Cartesian system \((X', Y', Z')\) is defined. In this *shell coordinate* system, the \((X', Y')\) axes are in the directions of the tangents to the grid lines, the \(Z'\) axis is chosen to form a right-handed orthogonal system. Loads, displacements, stiffener spacings, and eccentricities are generally defined in terms of these local coordinates. In computer output, primes are not used.
In the manual as well as in the computer output, reference is made to an inside and an outside surface. The outside surface is always defined as the shell wall surface corresponding to the largest positive or least negative value of \( Z' \).

If the shell wall or layer of the shell wall is orthotropic with respect to some direction, then the stiffness properties are most easily defined in an orthogonal system \((\phi_1, \phi_2)\) in the plane of the shell. This is referred to as the *material coordinate* system. Internally in the program the properties of the shell wall (or of its layers) are transformed from the \((\phi_1, \phi_2)\) system to the \((X', Y')\) system. See Figure 5.7 on page 5-133.

Stiffeners are structural elements that are attached to the shell surface. Their cross-sections cannot deform or warp. At times in the manual, stiffeners following the grid lines corresponding to constant \( Y \)-values are referred to as stringers, while those on constant \( X \)-values are called rings. The *cross-section coordinate* system \((\bar{y}, \bar{z})\) is a special local Cartesian system used for the definition of cross-section properties. Stiffeners can either be considered as discrete, in which case they are defined one by one, or their contribution to the shell wall stiffness can be “smeared” over the shell surface. Smeared stiffeners are used for convenience (to reduce input requirements) or in order to suppress local deformation between stiffeners. The use of smeared stiffeners also reduces the demand on limited computer resources.

The *element coordinate* system \((x', y', z')\) is used for local Cartesian systems defining the orientation of finite elements (element unit). **STAGS** also uses such a frame for every element, but except for stress output requested in the element frame, these coordinates can be ignored by the user for the standard geometries.

A shell unit may be mapped onto a rectangular domain (the mapping is analytic if the surface coordinates \((X, Y)\) are used). Consequently, it can always be considered as a surface with four sides (see Figure 4.3, on the following page). These sides are referred to in the user instructions by the integers 1, 2, 3, 4. The node points are arranged in a rectangular numbering system, so that each is given a row and a column index. Side 1 is always the side on which the node points have the row number 1; Side 4 is the side on which the column numbers are 1. If \((X, Y)\) coordinates are used, Side 1 is the side on which the \( X \) is zero, Side 4 is the side on which \( Y \) is zero. Side 3 is opposite Side 1, and Side 2 is opposite Side 4. The sides are arranged in clockwise order. As examples, a rectangular plate and a cylindrical shell are shown in Figure 4.1.
4.3 The Element Unit

In preparation.

4.4 Assembled Structures

In preparation.

4.5 Boundary Conditions

In preparation.
4.6 Loads

**ROUGH DRAFT**

![Diagram of load types](image)

<table>
<thead>
<tr>
<th>Line loads</th>
<th>Surface traction</th>
<th>Live pressure</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ F_u F_v F_w ] [ M_u M_v M_w ]</td>
<td>[ F_u F_v F_w ]</td>
<td>[ p - force/area ]</td>
</tr>
</tbody>
</table>

- \[ F_i \] = force/length
- \[ M_i \] = moment/length

Figure 4.4 STAGS distributed loading types.
Table 4.2  Shell unit distributed loading.

<table>
<thead>
<tr>
<th>coordinate system</th>
<th>line loads</th>
<th>traction</th>
<th>live pressure</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (x_e, y_e, z_e) )</td>
<td>✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>( (X', Y', Z') )</td>
<td>✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>( z' )</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>

Table 4.3  Element unit distributed loading.

<table>
<thead>
<tr>
<th>coordinate system</th>
<th>line loads</th>
<th>traction</th>
<th>live pressure</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (x_e, y_e, z_e) )</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( (x_s, y_s, z_s) )</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( z' )</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>
4.7 Summary of Modeling Techniques

Before preparation of input data, the user should carefully consider the modeling of the structure and the computation strategy to be followed. The choice of analysis type and problems connected with modeling and strategy are discussed in Chapter 11. Some control and summary data occurring early in the input can be defined only after the discretization and the computational strategy have been determined.

Execution of the STAGS program always proceeds in two distinct steps, the first being the model setup phase (S1) and the second being the analysis phase (S2). In previous versions of the program, all data were read in at the beginning, including solution strategy and control data. This was true even though in principle it was not necessary to execute the model setup step again for changes in the solution parameters. These data were always available at the beginning of the analysis phase. STAGS now provides two sets of input data: one for model setup (see Chapters 5–10) and one for solution control (see Chapter 11). To maintain upward compatibility of STAGS input run streams, solution control records that previously appeared in the initial input have been preserved. These records will be repeated in the input to the analysis program, and include records B-1, C-1, and all D and E records. The advantage of having two separate input files is that it permits a user to restart or change strategies using previously-saved model data, making additional executions of the model setup step unnecessary. The very brief input stream for the analysis phase is described in Chapter 11.

The model setup is defined in terms of shell units and element units:

- Shell units. These modeling units are typically thin plate or shell components—but they may also be single- or multi-layered sandwich or solid components. Their discretization is defined by reference to a numbering system based on rows and columns. Different options are available for discretization.

- Element units. These modeling units consist of thin, shell-type and/or thicker sandwich and/or solid-type finite elements defined either individually on regular data records or in batches through user-written subroutines. Such batches can be used, for example, for definition of a shell segment with a discretization that does not fit into the row and column scheme required for a shell unit. Finite element models developed using modeling pre-processor software systems such as PATRAN or I-DEAS typically generate element unit input data.

For a better overview, the different logical records have been divided into groups so that the contents of the records within the same group contain information of similar type. The groups of data in the model setup step are contained in the INP file and include the following groups of input records:
• Title record. Title of the model and analysis problem.

• Summary and Control Parameters. Various model controls—including the number of modeling units, the number of materials, the types of material models, and so forth.

• Discretization and Connectivity Summary. These records define the shell unit discretization in terms of the number of rows and columns in each shell unit, the number of nodes and elements in each element unit, connectivities among the modeling units, crack modeling, MPC definitions, and partial compatibility constraints.

• Data Tables. These include the Material Table, the Generic Constitutive Processor Table, the Beam Cross-Section Table, the Shell Wall Property Table, and the User Parameter Table:
  • Material Table. This table defines elastic properties and, if necessary, stress-strain curves for a number of materials.
  • Generic Constitutive Processor Table. This table defines material properties, stress-strain curves (as necessary), and shell and solid fabrication types—with STAGS’ Generic Constitutive Processor (GCP) functionality.
  • Beam Cross-Section Table. This table defines cross-sections and (with reference to the Material Tables) materials of beams and stiffeners.
  • Shell Wall Property Table. This table defines geometry and (with reference to the Material Table) material of shell wall configurations.
  • User Parameter Table. This table defines constants (integers) and floating point numbers, that are read in for possible use in user-written subroutines.

Next follows detailed information about all modeling units. For each modeling unit the following groups of input records are read in order.

• Geometry and Material
• Discretization
• Discrete Stiffeners
• Boundary Conditions
• Loads
• Output Control

The input for one unit is completed before any entry is made for the following unit. Shell and element units are numbered in the order in which they are specified and are referred to in the output by this “unit” number. Properties are defined by reference to the Data Tables whenever appropriate.
The last groups of input records are referred to as Element Units. In contrast to shell units, all available finite element types can be intermingled within the same unit.

Detailed information is given in Chapters 7–10 and in the STAGS Elements Manual about the properties of finite elements in the STAGS element library. The available elements include:

- Mount and Fastener Elements (general nonlinear springs)
- General Beams (torsional and axial bars)
- Rigid-Link and Soft-Link Beam Elements
- Triangular Plate/Shell Elements
- Quadrilateral Plate/Shell Elements
- Quadrilateral Plate/Shell Mesh Transition Elements
- Solid Elements and Shell/Solid Sandwich Elements
- Solid and Sandwich Mesh Transition Elements
- Surface/Surface, Point/Surface, and Line Contact Elements

Some considerations and restrictions should be noted:

- The E210 beam element, in STAGS, is compatible with the E320 triangular shell element and with the E410 quadrilateral shell element. It is not compatible with higher-order elements.
- The E330 triangular shell element, in STAGS, is not compatible with other the shell elements.
- GCP functionality, in the current version of STAGS, is available with the E330, E410 and E480 shell elements and with all solid and sandwich elements. GCP functionality has not been implemented for the E210 beam element or for the E320 shell element.
- The Poisson’s ratio parameter specified on the I-2 record is the minor Poisson’s ratio. The Poisson’s ratio required for GCP input is the major Poisson’s ratio.
- Material density, in the current version of STAGS, is defined as weight density in the traditional (I-record) specifications—and as mass density within the GCP.
- The plasticity option can only be used with an initially isotropic material. It cannot be used with material properties that vary with the surface coordinates. The stress-strain curve must be independent of temperature.
- Bifurcation buckling and small vibration analysis may not include plasticity.
- Progressive failure analysis, in STAGS, is supported only under the GCP and is not active for linear stress and/or eigenvalue analyses.
- The default definition of the gravitational acceleration constant \((g)\) is 386.1 in/sec\(^2\).

- When the very first character in any record of a STAGS input file is an upper case “C” or a “$” sign, the entire record is treated as a comment. When the very first character is a lower case “c” the record is treated as input data.

Experienced users may find it convenient to utilize the Input Record Catalog, Appendix B.

The groups of data in the solution control step are contained in the BIN file, which contains the following groups of input records:

- Title record. Title of the model and analysis problem.

- Summary and Control Parameters. Specifies analysis type, equation solver, corotation options, higher-order imperfection flag, output options, and so forth.

- Computational Strategy Parameters. These records specify load factors, solution strategies, eigenvalue analysis control parameters, time integration parameters, convergence controls, and so on.
STAGS user input consists of two text files, the \textit{INP} file and the \textit{BIN} file (see Chapter 17 “Input/Output Files” for a summary description of all I/O files). The \textit{INP} file is described in Chapters 5–10, and the \textit{BIN} file is described in Chapter 11. Each of these two input files consists of a sequence of input records, which have a required order of appearance. Moreover, data have a required order of entry on each record. Chapters 5–11 describe the input records sequentially, in the required order of appearance in the \textit{INP} and \textit{BIN} files. Each record description indicates the required order of input data comprising that record. See Appendix B “Input Record Catalog” for a summary of input records.

Conventions used in describing input records are presented in Section 5.1 “Conventions Used in Input-Record Descriptions”. An example input-record description, for a typical \textit{INP-file} record, is presented in 5.2 “Example Input-Record Description”. Conventions governing required input-data format for most of the \textit{INP-file} records and for all of the \textit{BIN-file} records are presented in Section 5.3 “Input Format Conventions”. Conventions used in defining \textit{User elements} are described in Chapter 13.

\textbf{Special Fonts}

- \textbf{Record-ID} \hspace{1cm} See \textit{Record ID and title}, on the following page
- \textbf{DATA} \hspace{1cm} See \textit{Data summary}, on the following page
5.1 Conventions Used in Input-Record Descriptions

Conventions used in input-record descriptions in this document are explained via reference to an example — “I-2 Material Elastic Properties” on page 5-59. Each record description consists of five sections.

Record ID and title

The first line of each record description contains the record ID followed by the record title. In the example “I-2 Material Elastic Properties”, the record ID is “I-2” and the record title is “Material Elastic Properties.” Records are sometimes identified by referring to their record ID’s, which appear in a special font. For example, “see I-2” and “(I-2)” are shorthand notations for “see the documentation for the I-2 record.”

Discussion

A short discussion regarding the data that are entered on the record follows the record ID and title line.

Data summary

This is a one-line summary of data that are entered on the record, indicating the required order of entry. The I-2 record contains seven data items. Each input datum is identified by a datum name. Data names appear in uppercase in a special font, as in RHO. The notation “RHO (I-2)” is shorthand for “the datum RHO on the I-2 record”. If it is clear that the I-2 record is indicated, then RHO may appear without the corresponding record ID.

Data description

Each datum is described in detail, in the order it is entered on the record.
Where to go from here

Input records have a required, application-dependent, order of appearance. Some records are always included, while others are required only under certain circumstances. The final section of each record description contains logic, written in pseudo-code, which indicates the next record to appear in the input file. This section is marked with a compass symbol, shown in the box to the right of this paragraph, suggesting its use in navigating through the hazardous waters of STAGS input data.

5.2 Example Input-Record Description

In the following, each of the five record description parts is identified for the chosen example, “I-2 Material Elastic Properties” on page 5-59.

Record ID and title:

I-2 Material Elastic Properties

Discussion:

In plate or shell analysis the plane stress assumption is used; the transverse normal stress is assumed to be zero. In that case, the elastic properties are defined by a matrix $C$ such that

$$\begin{pmatrix} E_1 & G & \nu_{12} \\ G & E_2 & \nu_{21} \\ \nu_{12} & \nu_{21} & \nu_{22} \end{pmatrix}$$

If the material is isotropic, it is not necessary to input all the four material constants. When the shear modulus is set to zero, it is assumed that the material is isotropic, i.e., that $E_2 = E_1$ and $G = E_1/\{2(1 + \nu_{12})\}$. If a material with a zero shear modulus must be defined, input a very small value for $G$. If the coefficient of thermal expansion in the $\phi_2$ direction is set equal to zero, it is assumed that $\alpha_1 = \alpha_2$ ($A_1 = A_2$).
Data summary:

- \( E_1 \)  
- \( U_{12} \)  
- \( G \)  
- \( \rho \)  
- \( A_1 \)  
- \( E_2 \)  
- \( A_2 \)

Data description:

- \( E_1 \) elastic modulus in \( \phi_1 \) direction
- \( \cdots \)
- \( A_2 \) coefficient of thermal expansion in \( \phi_2 \) direction, \( \alpha_2 \). Setting \( A_2 = 0 \) causes enforcement of the relationship \( \alpha_2 = \alpha_1 = \alpha \).

Where to go from here:

- \( \text{NESP} \) (I-1) number of points on the \((\sigma, \varepsilon)\) curve
  
  if \( \text{NESP} > 0 \) then \text{go to I-3}
  
  else \text{follow instructions at end of I-3a}

5.3 Input Format Conventions

A free-form input record contains a number of data fields—each of which is separated from its neighbor(s) by one of several data terminators—and optional comments that may be placed after the end of input data and are ignored by the program. Most of the input records for STAGS INP and BIN files contain only numerical data, consisting of one or more integer-type and/or floating-point-type data fields. Each numerical data field specifies zero or more integer or floating point data values. The zero or more part of this statement is describe below, after the following bulleted notes about numerical data fields:

- Unless otherwise noted, an entry on a data record is an integer if the variable name starts with I–N. Otherwise it is to be treated as a floating-point number.
- Integer field widths cannot exceed 10 characters, including the optional plus or minus sign. Leading blanks and a terminating character are not counted.
• Floating-point numbers must contain a decimal point; an exponent is optional. The exponent may be in any form allowed by regular FORTRAN “E” type format. The following data forms are all equivalent:

\[1450.\quad 1.45E+03\quad .145+4\quad 1.45E3\]

• Floating-point decimal field widths are unlimited, however floating-point data are truncated to about 7 or 8 decimal digits.

• A numerical data field may begin with any number of blanks (which are ignored) and is terminated with one of the following characters:

```
    " "  blank
    ","  comma
    "/"  slash
    "$"  dollar sign
```

or by the end of STAGS’ record-input character-string buffer, which can accommodate up to (but not more than) 255 characters.

• **Blanks** are ignored everywhere on records containing numerical data, except when they occur between two numerical data fields; then the first field is terminated by the blank. Thus input integers or floating-point numbers may never contain embedded blanks.

**WARNING:** For most of the INP file (see Chapter 9 for exceptions to this) and for all of the BIN file, a totally blank line is not ignored; it produces a single logical record.

• A **comma** is normally used to terminate a data field when the logical record contains additional data. Hence, when the last data field on one line is terminated with a comma, the comma functions as a continuation character and indicates that the logical record continues with the next input line. Otherwise, the logical record is terminated when the end of data on the line is reached. Successive commas on a line may be used to generate blank values of integer items in a list. For example, “20,,10” is equivalent to “20, 0, 0, 10“, which is equivalent to “20 0 0 10”.

• A **slash** terminates a logical record, and means that any following data field begins a new logical record. Thus, several logical records may be constructed with one input line (whereas with the use of commas, several input records might comprise a single logical record). A slash following a comma prevents continuation and the comma then functions as a simple terminator.

• A **dollar sign** (or the “C” character) in column 1 indicates that the entire record is a comment. A comment record may appear anywhere in an input file, except for the first record, which is taken to be the title “card” (see “A-1 Case Title" on page 5-7). A dollar sign in any other column signals the end of data on the record and means that the remaining information on the line is ignored. Space following the dollar sign may
be used for comments. A dollar sign may also be used when the data field on a line is terminated by a comma, *i.e.*, when the logical record continues on the following input line.

The current version of **STAGS** recognizes the \([N*]V\) construct for data-field specification, where the brackets surrounding the \(N^*\) prefix indicate that it is *optional*, and where \(V\) represents a specific integer or floating-point value. Omission of the \(N^*\) prefix specifies a single instance of \(V\) for the data field in question. Use of the \(N^*\) prefix, with \(N\) a non-negative integer, makes **STAGS** interpret \(N^*V\) as \(N\) successive instances of the specified value \(V\). For example, **STAGS** interprets “2*33.” as “33. 33.” and “4*20” as “20 20 20 20”.

Some **STAGS** input records in the **INP** file begin with a *character-type* command entry that is (usually) followed by one or more *integer-type* and/or *floating-point-type* numeric data entries. **STAGS** recognizes the \([N^*]V\) construct for character-type information, too; but the current version of **STAGS** does not expect more than one consecutive character-type data field.

### 5.4 Summary and Control Parameters

The first group of input records consists of the A and B records.

The first two, A-1 and B-1, appear in both the **INP** (model input) file and the **BIN** (solution input) file. These are the only two records that appear in both input files.

A-1 is the Case Title record, which contains a character string strictly for the user’s convenience. B-1 is the Analysis Type Definition record.

The **INP** B-1 record is *not* interchangeable with the **BIN** B-1 record. They contain different input data altogether.
A-1 Case Title

The case title is read on the first line and may contain any alphanumeric characters. This text is printed at the beginning of the output for the case and for identification on any disk file saved for possible restart. Subsequently any number of comment records can be added provided they begin with a “$” in column 1. Comments can also be included at the end of a data line—a “$” terminating data, and the comment following. A list of the complete input file, including any comment records, is printed at the beginning of all text output files. The user is urged to use this record as a way to document the analysis.

COMMENT

COMMENT case title

go to B-1
B-1 Analysis Type Definition

This is a revised version of the B-1 record found in earlier versions of STAGS. Note that the BIN (solution input) file also includes a B-1 record, but that it contains data that are different than those found here. The B-1 records in the INP & BIN files are not interchangeable.

The B-1 record in the INP file is not interchangeable with the B-1 record in the BIN file. They contain different input data altogether.

<table>
<thead>
<tr>
<th>IGRAV</th>
<th>gravitational constant, ( g ), definition option; IGRAV is needed for mass computation, since density (RHO, l-2) and nodal mass (GM, Q-4 and U-4) are defined using units of force rather than mass.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICHECK</td>
<td>0 – normal execution</td>
</tr>
<tr>
<td></td>
<td>1 – quick execution for model verification only</td>
</tr>
<tr>
<td>ILIST</td>
<td>data printout option</td>
</tr>
<tr>
<td></td>
<td>0 – normal printout</td>
</tr>
<tr>
<td></td>
<td>1 – full printout</td>
</tr>
<tr>
<td>INCBC</td>
<td>incremental boundary condition (BC) flag. (See 6.4 “Boundary Conditions” on page 6-57.)</td>
</tr>
<tr>
<td></td>
<td>0 – incremental BC are not used</td>
</tr>
<tr>
<td></td>
<td>1 – incremental BC are included</td>
</tr>
</tbody>
</table>

When incremental BC are included (INCBC = 1), the default condition is that they are the same as basic BC. Default incremental BC may be selectively overridden in the following ways:

Shell Units: IBOND (P-1)

Shell Units and Element Units: load system B specified displacements (LT = -1, Q-3/U-3) are interpreted as incremental BC.
In this situation, the value of the displacement (P, Q-3/U-3) is ignored, and a dof constraint is imposed (i.e., specified zero).
NRUNIT  model plot control variable
       0 – plot the entire model
       >0 – plot units listed in B-1a

NROTS  plot rotation specification variable
       0 – use the default plot orientation
       >0 – use sequence of rotations given in B-1b list

KDEV  plot device indicator
       0 – use the pdf (Acrobat) format for model plot
       1 – use the PostScript file format for model plot
       -1 – generate no model plot

NRUNIT (B-1)  model plot control variable
NROTS (B-1)  plot rotation specification variable

if    (NRUNIT > 0) then  go to  B-1a
elseif (NROTS > 0) then  go to  B-1b
else  go to  B-2
B-1a Model Unit List

List of units to be included in the model plot.

(IUNIS(i), i = 1, NRUNIT)

IUNIS(i)    unit number to be included in the model plot.

NROTS (B-1) plot rotation specification variable

if (NROTS > 0) then go to B-1b
else            go to B-2
**B-1b Sequence of Model Rotations**

Sequence of rotations to be imposed on model plot. Record B-1b is repeated \textbf{NROTS} times. Each rotation is in the \textit{global body system}, which is initially coincident with the \((x_g, y_g, z_g)\) global system. The current global body system is the result of all previous rotations; this corresponds to the “relative rotation” option in \texttt{PATRAN}. A particular application of this principle is the situation where you wish to alter the orientation of the model plot after having chosen the default value \texttt{NROTS}=0. If \texttt{NROTS} =0, \texttt{STAGS} will choose an orientation based on the model dimensions, with the rotation angles and axes displayed on the model plot. If you want to alter the orientation based on the picture, you must first input the sequence (of up to three rotations) printed by \texttt{STAGS}, after which any number of additional rotations may be specified. An example of this is the situation where you like the \texttt{STAGS} default orientation but wish to view the other side. Suppose in this case, you want to rotate 180 degrees about the current \textit{x} axis, and suppose \texttt{STAGS} has printed 93 degrees about \textit{x}, -26 degrees about \textit{y}, and 15 degrees about \textit{z}; then you would input \texttt{NROTS}=4, followed by the following four B-1b records:

1  93.
2  -26.
3   15.
1  180.

Alternatively, these four model rotations can be specified on a single line of text using the logical record separator; that is,

1  93. / 2 -26. / 3 15. / 1 180.$ B-1b Model Rotations

---

**IROT ROT**

- **IROT**
  - rotation axis.
  - 1 – rotation about the global body \textit{x} axis
  - 2 – rotation about the global body \textit{y} axis
  - 3 – rotation about the global body \textit{z} axis

- **ROT**
  - rotation angle (degrees)

---

*go to B-2*
B-2 General Model Summary

This record consists of an overall summary of model characteristics, such as the number of shell and element units, the number of units with stiffeners, what kind of constraints are required, and whether there are initial imperfections.

Very small initial geometric imperfections must sometimes be used to trigger deformation in the buckling mode. Imperfections of somewhat larger amplitudes must be included in the analysis model if the degree of imperfection sensitivity of the collapse load is to be established.

A number of different ways to include geometric imperfections is available. One of the options is to use a linear combination of buckling modes computed in an earlier execution of the program to define the shape of the imperfections. If this option is employed, the user must specify on the B-2 record the number of buckling modes $\text{nimpfs}$ to be read from the $\text{IMP}$ file. The amplitudes of the imperfection modes are read on the B-5 record. Other options to define geometric imperfections are described on the M-5 record.

STAGS also permits the user to impose a $\text{moving plane}$ boundary condition. The moving plane acts in the same way as an ordinary symmetry condition except that the plane of symmetry is allowed to translate and rotate as part of the system response. Thus, all translations along the specified boundary are restricted to lie in a plane, and only rotations normal to the plane are permitted. The program automatically introduces the appropriate Lagrange constraints in the proper order to enforce the boundary condition. The method is in force for all solution options, including the default nonlinear large rotation (corotational) algorithm. STAGS treats the moving plane boundary as new line elements of special type $\text{E250}$, which are treated just like the other beams in the 200 series. These “beams” can be invoked either directly in the element units, or as special rings or stringers to be placed along any row or column line in a shell unit that $\text{initially}$ satisfies the planar condition. Users should visualize the new input as a new kind of stiffener of special cross section -1 (reserved internally by STAGS) that cannot deform out of plane (large area moment), cannot twist (large torsional constant), but is free to deform within the plane (small area moment). It should be noted that the code enforces the boundary exactly, and does not actually put cross-section properties on this boundary. The user must increment $\text{nstfs}$ by one for each planar boundary used (below).
NUNITS NUNITE NSTFS NINTS NPATS NCONST NIMPFS INERT NINSR NPATX NSTIFS

NUNITS  number of shell units

NUNITE  number of element units; if NUNITE > 0, include an H-1 record for each element unit

NSTFS  number of shell units with discrete stiffeners. Do not forget the F-2 record. User must also supply a stiffener for each moving plane boundary desired. Moving plane boundaries can, of course, be mixed with ordinary stiffeners, including real stiffeners on the boundary itself.

NINTS  number of (G-1-record) specifications of connections between shell units. Each such specification establishes displacement compatibility for two shell unit boundary lines. G-1 can also be used to define the internal connection in closed shell units.

NPATS  number of records (G-2) used to define partial compatibility of displacements at specified node pairs (all types of units); also see NPATX, below

NCONST  number of constraint relations defined in user-written subroutine UCONST or on G-4 records. If there is a subroutine UCONST, NCONST equals the number of times CONSTR is called in UCONST. For bifurcation buckling or for small vibrations, constraints defined by UCONST apply both to the stress state and the incremental displacement. If NCONST < 0, |NCONST| constraints will be read on the G-3 and G-4 records called “regular constraints.”

NIMPFS  number of buckling modes read from the IMP file to be used to generate imperfections. If NIMPFS > 0, scale factors must be provided on the B-5 record.

INERT  number of inertial-load records (B-6)

NINSR  number of inserted node/element sets (G-5–G-7) defining cracks

NPATX  number of records (G-2c) used to define partial compatibility of displacements at specified node pairs in multi-layered shell and/or element units; also see NPATS, above

NSTIFS  number of linear-stiffness-matrix contributions (W-1)

goto B-3
B-3 Data Tables Summary

This record specifies the number of library entries for materials, cross sections, wall fabrications, user input, and mount elements. It also specifies whether or not STAGS’ version of the Generic Constitutive Processor (GCP) is to be used for material-property and wall-fabrication information, in addition to or instead of the above-mentioned libraries (the traditional STAGS approach). Whenever data in a library entry or in the GCP system are needed, the appropriate library is referenced by an integer identifier, as explained in records I, J and K. The User Parameter records L provide a means to pass user-defined data to user-written subroutines.

<table>
<thead>
<tr>
<th>NTAM</th>
<th>NTAB</th>
<th>NTAW</th>
<th>NTAP</th>
<th>NTAMT</th>
<th>NGCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of entries (materials) in the Material Table (I-1).</td>
<td>number of entries (cross-sections of beams or stiffeners) in the Beam Cross-Section Table (J-1).</td>
<td>number of entries (shell wall types) in the Shell Wall Property Table (K-1).</td>
<td>0 – User Parameters are not included (L-1).</td>
<td>1 – User Parameters are included (L-1).</td>
<td>GCP-utilization flag:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 – The GCP system will not be used</td>
<td>1 – The GCP system will be used</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IGRAV (B-1) gravitational constant, \( g \), definition option
NIMPFS (B-2) number of buckling modes read from the IMP file
INERT (B-2) number of inertial-load records
NUNITS (B-2) number of shell units

if (IGRAV = 1) then go to B-4
elseif (NIMPFS > 0) then go to B-5
elseif (INERT > 0) then go to B-6
elseif (NUNITS > 0) then go to F-1
else follow instructions at end of G-1
B-4 Gravitational Acceleration

This record modifies the gravitation constant if the units used are other than inches and seconds. It is included only if $\text{IGRAV} = 1$ (B-1).

---

**GRAV**

*GRAV* gravitational acceleration (*e.g.*, $g = 9.80665 \text{ m/s}^2$)

- **NIMPFS** (B-2) number of buckling modes read from the *IMP* file
- **INERT** (B-2) number of inertial-load records
- **NUNITS** (B-2) number of shell units

if $\text{NIMPFS} > 0$ then *go to B-5*
elseif $\text{INERT} > 0$ then *go to B-6*
elseif $\text{NUNITS} > 0$ then *go to F-1*
else *follow instructions at end of G-1*
B-5 Buckling Mode Imperfections

These records are included only if $\text{NIMPFS} > 0$ (B-2). STAGS allows the user to include imperfections from a number of eigenvalue executions of the same case. For example, one may have initially performed a linear bifurcation analysis, followed by a nonlinear collapse analysis, followed by a mode computed from the nonlinear stress state. Another execution could be run from the beginning, including all the modes accumulated so far. For each eigenvalue execution, STAGS produces two files, IMP and EGV, that contain all the modes computed during the entire history of the case. These modes are identified by the load step number, IMSTEP; the mode number, IMMODE; and the eigenvalue-execution ID number, IMRUN. These mode-identifying data are found in the OUT2 files corresponding to the specified eigenvalue executions. The user can generate any linear combination of these modes by specifying their amplitudes (remember, for eigenmodes computed by STAGS, the largest translational component is normalized to unity). These imperfections are combined with any imperfections generated by M-6 records or by user-written subroutine DIMP. The eigenvectors identified on the B-5 record are read from the EGV/IMP file set; see Chapter 17 “Input/Output Files”. Note that the buckling mode amplitude $WIMPFA$ multiplies all translational components in the eigenvector—not just the transverse component.

This record is repeated $\text{NIMPFS}$ times.

<table>
<thead>
<tr>
<th>WIMPFA</th>
<th>IMSTEP</th>
<th>IMMODE</th>
<th>IMRUN</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIMPFA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMSTEP</td>
<td>load step number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMMODE</td>
<td>mode number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMRUN</td>
<td>eigenvalue-execution ID number</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$\text{NIMPFS}$ (B-2) number of buckling modes read from the IMP file
$\text{INERT}$ (B-2) number of inertial-load records
$\text{NUNITS}$ (B-2) number of shell units

if (NIMPFS modes have been defined) then
  if (INERT > 0) then go to B-6
  elseif (NUNITS > 0) then go to F-1
  else
    follow instructions at end of G-1
else continue defining B-5
B-6 Inertial Loads Summary

This record is included only for inertial loading, \( \text{INERT} > 0 \) (B-2). Each of the \( \text{INERT} \) inertial load records is defined via a B-6–B-6d sequence and may be independently added to Load System A or to Load System B. Inertial loading constitutes part of the base loads, which are scaled by the load factors \( P_A \) and \( P_B \) (see Section 6.5).

A general inertial field which acts over the entire structure is achieved by prescribing:

- components of an acceleration vector \( (a_o) \) acting in the global coordinate directions \( (x_g, y_g, z_g) \)
- components of an angular velocity vector \( (\omega_o) \), acting about the axes of a coordinate system whose directions are parallel to \( (x_g, y_g, z_g) \) and whose origin is specified by the position vector \( (x_o) \)
- components of an angular acceleration vector \( (\alpha_o) \), acting about the axes of the coordinate system described above for \( (\omega_o) \)

The resulting inertial loads are computed on a nodal basis as

\[
f_i = m_i (-a_i)
\]

where \( f_i \) is the inertial force vector which acts on node \( i \) due to the product of \( m_i \), the “lumped mass” at node \( i \), and \( a_i \), the acceleration at node \( i \), computed as the sum of gravitational acceleration and centripetal acceleration according to

\[
a_i = a_o + \omega_o \times (\omega_o \times r_i) + \alpha_o \times r_i
\]

where \( a_o \) is the input acceleration vector, \( \omega_o \) is the input angular velocity vector, \( \alpha_o \) is the input angular acceleration vector, and \( r_i \) is the relative location of node \( i \), computed as

\[
r_i = x_i - x_o
\]

where \( x_o \) is the vector which positions the origin of the coordinate system used to compute centrifugal loading, and \( x_i \) contains the \( (x_g, y_g, z_g) \) coordinates of node \( i \).

Notice the minus sign in the expression for \( f_i \), which expresses the principle that inertial forces oppose acceleration. For example, to impose gravity loading on a structure with the negative \( z_g \) axis pointing towards the center of the earth, specify \( a_o = (0, 0, g) \), \( \omega_o = 0 \), and \( \alpha_o = 0 \).
Model Input Summary and Control Parameters

**ISYS IA IOM IAL IOPT**

**ISYS**
load system option

1  –  Load system A
2  –  Load system B

**IA**
acceleration ($a_o$) option

0  –  acceleration vector omitted
1  –  acceleration vector specified on B-6a

**IOM**
angular velocity ($\omega_o$) option

0  –  angular velocity vector omitted
1  –  angular velocity vector specified on B-6b

**IAL**
angular acceleration ($\alpha_o$) option

0  –  angular acceleration vector omitted
1  –  angular acceleration vector specified on B-6c

**IOPT**
coordinate system option for angular velocity/acceleration

0  –  $\omega_o$, $\alpha_o$ are about the $(x_g', y_g', z_g)$ system; $x_o = 0$
1  –  $\omega_o$, $\alpha_o$ are about a set of axes which are parallel to $(x_g', y_g', z_g)$ and centered at the *center of mass* for the structure; $x_o$ is computed by STAGS
2  –  $\omega_o$, $\alpha_o$ are about a set of axes which are parallel to $(x_g', y_g', z_g)$ and centered at a user-specified position; $x_o$ is input on B-6d

if (IA = 1) then  
go to B-6a
elseif (IOM = 1) then  
go to B-6b
elseif (IAL = 1) then  
go to B-6c
elseif (IOPT = 2) then  
go to B-6d
else     
follow instructions at end of B-6d
B-6a Inertial Loads — Acceleration

AX, AY, AZ  components of the acceleration vector \( \mathbf{a}_o \); units are \( \text{length/time}^2 \)
(e.g., \( m/s^2 \) or \( \text{ft/s}^2 \))

- **IOM** (B-6)  angular velocity \( (\omega_o) \) option
- **IAL** (B-6)  angular acceleration \( (\alpha_o) \) option
- **IOPT** (B-6)  coordinate system option for angular velocity/acceleration

if \( \text{IOM} = 1 \) then  \textit{go to B-6b}
elseif \( \text{IAL} = 1 \) then  \textit{go to B-6c}
elseif \( \text{IOPT} = 2 \) then  \textit{go to B-6d}
else  \textit{follow instructions at end of B-6d}
OMX, OMY, OMZ components of the angular velocity vector $\omega_o$; units are $\text{radians/time}$, $(\text{e.g.,}\ \text{radians/s})$

**IAL** (B-6) angular acceleration ($\alpha_o$) option

**IOPT** (B-6) coordinate system option for angular velocity/acceleration

if (IAL = 1) then go to B-6c
elseif (IOPT = 2) then go to B-6d
else follow instructions at end of B-6d
**B-6c Inertial Loads — Angular Acceleration**

**ALX, ALY, ALZ**

Components of the angular acceleration vector \( \alpha \); units are \( \text{radians/time}^2 \), 
\( (\text{e.g., radians/s}^2) \)

**IOPT (B-6)**

Coordinate system option for angular velocity/acceleration

if \( \text{IOPT} = 2 \) then go to B-6d
else follow instructions at end of B-6d
B-6d Inertial Loads — Position Vector

This record is included only when the user is specifying a position vector for angular velocity ($i_{OPT} = 2, B-6$).

- $X$, $Y$, $Z$ components of the position vector $x_o$

- **INERT (B-2)** number of inertial-load records
- **NUNITS (B-2)** number of shell units

\[
\begin{align*}
\text{if} & \quad (\text{INERT inertial-load records have been defined}) \quad \text{then} \\
\text{if} & \quad (\text{NUNITS} > 0) \quad \text{then} \quad \text{go to F-1} \\
& \quad \text{else} \quad \text{follow instructions at end of G-1} \\
& \quad \text{else} \quad \text{return to B-6}
\end{align*}
\]

5.5 Discretization and Connectivity Summary

The next group of records in the INP file provides a summary of the discretization of each of the modeling units and specifies connectivities among these units—including crack definitions (if any). Shell unit discretizations are defined via F-1 Grid Summary and F-2 Stiffener Summary records. Element unit discretizations are summarized on the H-1 Element Unit Summary record. Connectivities among the modeling units, partial compatibility constraints, and multi-point constraints are defined via G-1 through G-4 record groups. Crack definitions are specified via G-5, G-6 and G-7 records.
**F-1 Grid Summary**

The grid point summary record gives the number of rows and columns for each of the NUNITS (B-2) shell units. While in most cases, these coincide with the element edges, for nine-node shell elements with full-fledged midside nodes, every other gridline coincides with the element edges. Thus, for the nine-node E480 element, for example, the element spans three gridlines instead of two. That is, a grid specified by three rows and three columns (a 3x3 grid) accommodates a single 9-node quadrilateral element or four 4-node quadrilateral elements. NROWS and NCOLS must be odd to accommodate these elements. For ordinary quadrilateral shell elements, the number of elements $N_{er}$ along a row is

$$N_{er} = NCOLS - 1$$

while for nine-node shell elements it is

$$N_{er} = (NCOLS - 1)/2$$

with corresponding formulas for the number of elements along a column.

| NROWS(i) | number of rows for shell unit $i$ |
| NCOLS(i) | number of columns for shell unit $i$ |

**NOTE:** All of the data here are on a single logical record; when the user’s model has a large number of shell units, it is generally necessary to divide this logical record into two or more subrecords, each (except the last) of which is terminated with a record-continuation character (comma) (see 5.3 “Input Format Conventions” on page 5-4).

NSTFS (B-2) number of shell units with discrete stiffeners
NINTS (B-2) number of shell-unit-connection records required
NPATS (B-2) number of G-2 type partial-compatibility records required
NPATX (B-2) number of G-2c type partial-compatibility records required
NCONST(B-2) number of Lagrange-constraint equations
NINSR (B-2) number of inserted node/element sets defining cracks
NUNITE (B-2) number of element units

if (NSTFS > 0) then go to F-2
elseif (NINTS > 0) then go to G-1
elseif (NPATS > 0) then go to G-2
elseif (NPATX > 0) then go to G-2c
elseif (NCONST < 0) then go to G-3
elseif (NINSR > 0) then go to G-5
elseif (NUNITE > 0) then go to H-1
else follow instructions at end of H-1
**F-2 Stiffener Summary**

This record is included if and only if there are shell units with discrete stiffeners, *i.e.*, when $\text{NSTFS} > 0$ (B-2); this includes any *moving plane* boundary conditions. Ordinary stiffeners and moving plane boundaries can be mixed together in any order, and can lie on top of one another.

This record is repeated $\text{NSTFS}$ times.

### IUNIT NRGS NSTR

<table>
<thead>
<tr>
<th>IUNIT</th>
<th>shell unit number</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRGS</td>
<td>number of discrete stiffeners or moving plane boundaries in the row direction (constant $X$)</td>
</tr>
<tr>
<td>NSTR</td>
<td>number of discrete stiffeners or moving plane boundaries in the column direction (constant $Y$).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NSTFS</th>
<th>number of shell units with discrete stiffeners (B-2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NINTS</td>
<td>number of shell-unit connections (B-2)</td>
</tr>
<tr>
<td>NPATS</td>
<td>number of G-2 type partial-compatibility records required (B-2)</td>
</tr>
<tr>
<td>NPATX</td>
<td>number of G-2c type partial-compatibility records required (B-2)</td>
</tr>
<tr>
<td>NCONST</td>
<td>number of Lagrange-constraint equations (B-2)</td>
</tr>
<tr>
<td>NINSR</td>
<td>number of inserted node/element sets defining cracks (B-2)</td>
</tr>
<tr>
<td>NUNITE</td>
<td>number of element units (B-2)</td>
</tr>
</tbody>
</table>

if ($\text{NSTFS}$ F-2 records have been defined) then
if ($\text{NINTS} > 0$) then $\text{go to G-1}$
elseif ($\text{NPATS} > 0$) then $\text{go to G-2}$
elseif ($\text{NCONST} < 0$) then $\text{go to G-3}$
elseif ($\text{NINSR} > 0$) then $\text{go to G-5}$
elseif ($\text{NUNITE} > 0$) then $\text{go to H-1}$
else $\text{follow instructions at end of H-1}$
else $\text{continue defining F-2}$
G-1 Shell Unit Connections

Type G-1 record(s) are required when displacement compatibilities between pairs of shell unit boundary lines have been specified by defining $\text{NINTS}>0$ (B-2). Each G-1 record specifies one or (if the looping feature described below is used) more than one pair of shell-unit boundary lines for which displacement compatibilities are to be enforced.

Typically, G-1 records are used to join pairs of shell units that are constructed with thin-shell elements or to connect pairs of shell units that are constructed with solid elements. For some models, one or more G-1 records may be used to join pairs of shells where one is a thin-shell unit and the other is a solid-element (multi-layered) unit. Please see the note about that situation, later in this description.

In any event, the user should know that shell units are automatically numbered by the order in which they are defined. The boundary lines of the shell units are numbered as shown in Figure 6.6 on page 6-58. A G-1 shell unit connection record specifies compatibility between one boundary line on a shell unit $m$ and a second boundary line on a shell unit $n$. Shell unit $m$ must be chosen as the unit with the lower number of the two units involved in one G-1 record. G-1 can only be used to connect boundary lines of shell units. Connections can be defined at internal grid lines either by partial compatibility (G-2 and/or G-2c) records, with Lagrange constraints (G-3), or as user-written constraints.

If more than two shell units are connected at the same line, the $m$ unit on all of the corresponding G-1 records must be the same.

G-1 can be used to define a connection between two boundary lines on the same shell unit. This feature is used to define a closed shell. In that case, $\text{MUNIT} = \text{NUNIT}$; and $\text{MBOUND}, \text{NBOUND}$ are set to connect boundaries 1-to-3 or 2-to-4, as those are the only permissible ways to effect an intraunit connection. For example, a closed cylinder must be connected on the $0 = 0^\circ$ and $0 = 360^\circ$ boundaries (2-to-4); otherwise a “slit” cylinder results.

If $\text{NBOUND} < 0$, the juncture line on the unit $n$ is connected to the unit $m$ in reverse order. This option is used if the shell coordinate $X$ or $Y$ in the two connected units is increasing in opposite directions along the common boundary.

The user must insure that the grid lines in each unit are compatible along the line of juncture; this means that the number of rows or columns in one unit must equal either the number of rows or the number of columns in the other (F-1). For example, if a juncture line along a row on unit 1 is joined along a column in unit 2, the number of columns in unit 1 must be equal to the number of rows in unit 2 for a compatible fit. If compatibility conditions are not satisfied, a diagnostic message is printed.
Typically, a single G-1 record will be used to define each of the NINTS shell-unit connections for the problem at hand. For some configurations, however, it is convenient to use the G-1 looping feature described below to specify two or more connections with a single G-1 record; in this case, the number of G-1 records required will be less than NINTS (but must be sufficient to define all NINTS connections).

If one or the other (but not both) of the two units on a G-1 record identifies a multi-layered shell, it is

\[
\text{MUNIT MBOUND NUNIT NBOUND NDEFS INC1 INC2 INC3 INC4}
\]

- **MUNIT**: the magnitude of MUNIT identifies the lower-numbered shell unit \((m \leq n)\); see the note following this description for important information about the sign of MUNIT.
- **MBOUND**: boundary line number, for unit \(m\).
- **NUNIT**: the magnitude of NUNIT identifies the higher-numbered shell unit \((n \geq m)\); see the text above for important information about the sign of NUNIT.
- **NBOUND**: boundary line number, for unit \(n\).
- **NDEFS**: the number of shell-unit connections specified via the current G-1 record; STAGS sets NDEFS equal to unity if it is nonpositive or if it is omitted.
- **INC1**: incrementation parameter for MUNIT.
- **INC2**: incrementation parameter for MBOUND.
- **INC3**: incrementation parameter for NUNIT.
- **INC4**: incrementation parameter for NBOUND.

**Note:** A shell unit that is modeled with thin shell elements only needs a single layer of node points to define those elements. The shell may be a multi-layered shell in terms of its material fabrication properties; but it is a single-layered shell, geometrically. A shell unit that is modeled with sandwich or solid elements must have two or more layers of nodes in the thickness direction to accommodate those elements. The user typically establishes a “reference” layer of node points by specifying a geometry type (ISHELL) on an M-1 record and parameters associated with that type of shell on the M-2 companion for that M-1 record. When the user specifies the use of sandwich or solid elements for constructing the shell unit (on the N-1 record for that unit), STAGS starts with the reference layer of nodes (the reference grid) and generates additional nodal layers by “pulling” the reference layer through the thickness. The shell unit constructed by this process may be single layered with respect to its fabrication (regardless of the number of nodal layers required) but it is always multi-layered in a geometric sense. If one or the other (but not both) of the two units that are specified here identifies a multi-layered shell (with two or more nodal layers), the user must give a negative sign to the unit-specification variable for the multi-layered
unit (i.e., to MUNIT or NUNIT, depending on which of the two units is the multi-layered one). This tells STAGS to generate soft links (E121 elements) and Lagrange multipliers automatically, to enforce the displacement compatibilities that are needed to join pairs of shells when one of them is nodally multi-layered.

**Example 1:** the following G-1 record:

```
10 2 20 4 3 1 0 1 0 $ G-1 record
```

generates the same three shell-unit connections as the following three G-1 records:

```
10 2 20 4 $ G-1 record
11 2 21 4 $ G-1 record
12 2 22 4 $ G-1 record
```

**Example 2:** the following G-1 record:

```
-13 2 23 4 $ G-1 record
```

specifies a single connection in which a thin-shell unit is joined to a solid-shell unit.

- **NINTS** (B-2) number of shell-unit connections
- **NPATS** (B-2) number of G-2 type partial-compatibility records required
- **NPATX** (B-2) number of G-2c type partial-compatibility records required
- **NCONST** (B-2) number of Lagrange-constraint equations
- **NINSR** (B-2) number of inserted node/element sets defining cracks
- **NUNITE** (B-2) number of element units

if (NINTS shell unit connections have been defined) then
  if (NPATS > 0) then go to G-2
  elseif (NPATX > 0) then go to G-2c
  elseif (NCONST < 0) then go to G-3
  elseif (NINSR > 0) then go to G-5
  elseif (NUNITE > 0) then go to H-1
  else follow instructions at end of H-1
else continue defining G-1
**G-2 Partial Displacement Compatibility**

Record(s) of type G-2 are required when partial compatibility between displacements at particular nodes has been specified by defining $\text{NPATS} > 0$ (B-2). NPATS G-2 records must be included. Each G-2 record specifies one or more partial displacement compatibility conditions, depending on whether or not the G-2 record *exterior-looping* feature (described below) is used.

If G-2 records are used to connect single-layered shell units, $\text{IGLOBE}$ (M-1) must be 0, 3, 4, or 5. A single-layered-shell-unit displacement component is identified by its unit, row, column, and component (direction) numbers. A displacement component in element units is identified by node number ($\text{IR1}$), a zero for $\text{IC1}$, and a value for the direction ($\text{ID1}$). The first 8 items on each G-2 record define a pair of displacement components. The two units referenced may be the same. The first shell unit ($\text{IU1}$) must not be greater than the second ($\text{IU2}$).

The **zero rule** applies to the variables $\text{IR1}$, $\text{IC1}$, $\text{IR2}$, $\text{IC2}$: if any of these variables is set to zero in a shell unit, it means the compatibility condition applies to an entire row or column, respectively. For example, to enforce compatibility of the $u$ freedom along row 1 of unit 1, the single G-2 record shown below can be used:

```
1 1 1 1     1 1 0 1
```

When either a row (or column) entry is zero for both the master and slave sets of integers, a loop over all rows (or columns) is implied, generating $nrow$ (or $ncol$) separate constraints, where $nrow$ and $ncol$ are the number of rows and columns in the shell unit. For example, the single G-2 record

```
1 0 1 3     1 0 14 3
```

means that for each row, the $w$ freedom ($\text{ID1} = 3$) on column 14 is equated with the $w$ freedom on column 1. When this capability is used with two different shell units, the number of rows (columns) must of course be the same.

If the value of $\text{IR2}$ or $\text{IC2}$ (for the slave nodes) is negative, the compatibility between units $\text{IU2}$ and $\text{IU1}$ is enforced in reverse order (this is identical to what happens with the $\text{NBOUND}$ variable on G-1 records). This option is used if the shell coordinate $X$ or $Y$ in the two connected units is increasing in opposite directions along the common boundary.

Since no transformations are performed, the specified nodes should not lie on a boundary line involved in a shell unit connection (G-1) unless the two shell units have the same orientation with respect to the global coordinates.
A special option is provided for joining shell units using G-2 records. If both ID1 and ID2 are zero, then full compatibility is enforced between the nodes on IU1 and IU2. This means that the components on the node belonging to IU2 are first transformed into the local system in IU1 before the compatibility condition is enforced, which is the same thing that happens when shell units are joined using G-1 records. The zero rule also applies to the row and column indices here; thus, to join shell unit 2, row 1 to shell unit 1, row 13, the single record

```
1 13 0 0 2 1 0 0
```

suffices. Here, row 13 could be in the interior of the unit. The user must be very careful when using this option. First, he must insure the same row or column compatibility as required for the G-1 records; second, the user cannot set ID2 to zero to transform the coordinates in unit 2 to the master unit system when only a single component is specified in unit 1 (ID1 not zero); third, he must insure that the lines involved in the juncture actually coincide in space, since the code will not check for the physical coincidence of the juncture lines. The user can take advantage of IGLOBE options 3, 4, or 5 (M-1) to move the units around until they meet in the right place.

The next six items (ND1A through INC2) on the G-2 record are optional input variables allowing the user to specify full compatibility for a range of nodes. These data are ignored unless the following logical expression evaluates to ‘True’:

\[
\{(IR1 = 0 \quad or \quad IC1 = 0) \quad and \quad (IR2 = 0 \quad or \quad IC2 = 0)\}.
\]

These optional input variables provide a convenient (interior-looping) method for specifying compatibility between any portion of a row or column of one unit and a portion of a row or column of another unit. This type of input may also be used in element units provided the range of nodes has a fixed increment. For both shell and element units, increasing and decreasing ranges are allowed. If INC1 = 0 or INC2 = 0, an increment of +I is used for increasing ranges and -I for decreasing ranges.

Typically, one G-2 record will be used to define each of the NPATS partial displacement compatibility conditions for the problem at hand. For some configurations, however, it is convenient to use the G-2 exterior-looping feature described below to make two or more specifications with a single G-2 record. The final item (NDEFS) on the G-2 record is an optional control parameter that, when omitted or set equal to any integer less than or equal to unity, causes the G-2 record to generate a single (set of) partial displacement compatibility specification(s)— or, when NDEFS > 1, causes the G-2 record (with information on the G-2a record following it) to generate NDEFS sets of partial compatibility specifications (see the example following the G-2a description, below).
If the exterior-looping feature is not used on any of the G-2 records given, the NPATS G-2 records define NPATS partial displacement compatibilities; if the exterior-looping feature is used, the NPATS G-2 records given define more than NPATS partial displacement compatibility conditions.

<table>
<thead>
<tr>
<th>IU1</th>
<th>IR1</th>
<th>IC1</th>
<th>ID1</th>
<th>IU2</th>
<th>IR2</th>
<th>IC2</th>
<th>ID2</th>
<th>ND1A</th>
<th>ND1B</th>
<th>INC1</th>
<th>ND2A</th>
<th>ND2B</th>
<th>INC2</th>
<th>NDEFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>IU1</td>
<td>unit number of first node. IU1 = NUNITS + k indicates element unit k; IU1 must be the lower number (IU1 ≤ IU2); see CAUTION, below.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IR1</td>
<td>row number of first node or node number in element unit; if IR1 = 0, then either IR2 = 0 or IC2 = 0 must be set.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IC1</td>
<td>column number of first node. IC1 = 0 in element unit; if IC1 = 0, then either IR2 = 0 or IC2 = 0 must be set.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID1</td>
<td>Indicates the component involved at the first node; ID1 = 1, 2, 3, 4, 5 or 6 for u, v, w, ru, rv, rw, respectively; if ID1 = 0, then all components are involved, and ID2 = 0 must be set.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IU2</td>
<td>unit number of second node; IU2 = NUNITS + k for element unit k; IU2 must be the higher number (IU2 ≥ IU1); see CAUTION, below.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IR2</td>
<td>row number of second node or node number in element unit.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IC2</td>
<td>column number of second node; IC2 = 0 in element unit.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID2</td>
<td>indicates the component involved at the second node. ID2 = 1, 2, 3, 4, 5 or 6 for u, v, w, ru, rv, rw, respectively; if ID2 = 0, then the components are first transformed to the IU1 system before partial compatibility is enforced. Note: if ID2 = 0, then ID1 must also be 0.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ND1A</td>
<td>first node of: (row, if IR1 = 0) or (column, if IC1 = 0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ND1B</td>
<td>last node of: (row, if IR1 = 0) or (column, if IC1 = 0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC1</td>
<td>increment for nodes from ND1A to ND1B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ND2A</td>
<td>first node of: (row, if IR2 = 0) or (column, if IC2 = 0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ND2B</td>
<td>last node of: (row, if IR2 = 0) or (column, if IC2 = 0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC2</td>
<td>increment for nodes from ND2A to ND2B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDEFS</td>
<td>number of partial compatibility specifications generated via this G-2 record; set to unity by STAGS if nonpositive or omitted</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CAUTION: when different shell or element units are involved, the lower-numbered unit must be given first. This rule also applies to node numbers within a specific unit; the lower-numbered
node must be given first. In shell units, nodes are numbered in row-major order; *i.e.*, along row 1, starting at column 1; then along row 2, starting at column 1; *etc.*

if \((\text{NDEFS} > 0)\) then go to G-2a
else follow instructions at end of G-2a
G-2a Partial Compatibility Incrementations

A single record of type G-2a must be included immediately after each type G-2 record on which the NDEFS parameter is greater than unity. Eight incrementation variables are specified here for use with the G-2 record exterior-looping capabilities.

<table>
<thead>
<tr>
<th>JU1</th>
<th>JR1</th>
<th>JC1</th>
<th>JD1</th>
<th>JU2</th>
<th>JR2</th>
<th>JC2</th>
<th>JD2</th>
</tr>
</thead>
<tbody>
<tr>
<td>incrementation variable for use with IU1 (unit-specification) variable on the parent G-2 record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation variable for use with IR1 (on G-2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation variable for use with IC1 (on G-2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation variable for use with ID1 (on G-2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation variable for use with IU2 (on G-2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation variable for use with IR2 (on G-2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation variable for use with IC2 (on G-2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation variable for use with ID2 (on G-2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables may be negative, zero or positive.

**Exterior-looping example:** the following G-2/G-2a record combination:

```
63 2 0 4   71 2 0 4   1 7 1 1 7 1 3 $ G-2 record
0 0 0 1    0 0 0 1                    $ G-2a record
```

generates the same three partial compatibility definitions as the following three G-2 records:

```
63 2 0 4   71 2 0 4   1 7 1 1 7 1 $ G-2 record
63 2 0 5   71 2 0 5   1 7 1 1 7 1 $ G-2 record
63 2 0 6   71 2 0 6   1 7 1 1 7 1 $ G-2 record
```
NPATS (B-2) number of G-2 type partial-compatibility records required
NPATX (B-2) number of G-2c type partial-compatibility records required
NCONST (B-2) number of Lagrange-constraint equations
NINSR (B-2) number of inserted node/element sets defining cracks
NUNITE (B-2) number of element units

if  (NPATS G-2 records have been defined)  then
  if  (NPATX > 0)  then  go to  G-2c
  elseif  (NCONST < 0)  then  go to  G-3
  elseif  (NINSR > 0)  then  go to  G-5
  elseif  (NUNITE > 0)  then  go to  H-1
  else  follow instructions at end of  H-1
else  return to continue defining G-2
**G-2c Partial Displacement Compatibility**

Record pair(s) of type G-2c and G-2d are required when partial compatibility between displacements at particular nodes has been specified by defining \( \text{NPATX} > 0 \) (B-2): this protocol for specifying partial-compatibility conditions can be used with *single-layered* shell and/or element units; but it is most suitable for use with *multi-layered* shell and/or element units.

The term “single-layered” in this context refers to modeling units (or regions) where a single layer of node points (a two-dimensional grid) is needed to model the structure with a single layer of two-dimensional plate and/or shell elements. The term “multi-layered” refers, here, to units (or regions) where two or more layers of nodes are required to model a structural component with sandwich or solid elements that have two or more nodes in the “thickness” direction.

In any event, \( \text{NPATX} \) G-2c/G-2d record pairs must be included. Each G-2c record specifies one or more partial displacement compatibility conditions, depending on whether or not the *exterior-looping* feature is used on the G-2d record (described below). Each G-2c record must be followed immediately by a G-2d record.

If G-2c records are used to connect shell units, \( \text{IGLOBE} \) (M-1) must be 0, 3, 4, or 5. A displacement component in a single-layered shell unit is identified by its unit, row, column, shell-layer, and component (direction) numbers—with the shell-layer number set equal to zero. A displacement component in a multi-layered shell unit is identified by its unit, row, column, shell-layer, and component (direction) numbers—with the layer number set to a positive value that is less than or equal to the number of nodal layers in the unit. A displacement component in an element unit is identified by its node number \( \text{IR1} \), a zero for \( \text{IC1} \), a zero for \( \text{IL1} \), and a value for the direction \( \text{IDIR} \). The first 10 items on each G-2c record define a pair of displacement components. The two units referenced may be the same. The first unit number \( \text{IU1} \) must not be greater than the second \( \text{IU2} \).

The zero rule applies to the variables \( \text{IR1}, \text{IC1}, \text{IR2}, \text{IC2} \): if any of these variables is set to zero in a shell unit, it means the compatibility condition applies to an entire row or column, respectively. For example, to enforce compatibility of the \( u \) freedom along row 7 on nodal layer 5 of unit 2, the following G-2c record can be used:

\[
2 \ 7 \ 1 \ 1 \ 5 \ \ 2 \ 7 \ 0 \ 1 \ 5
\]

When either a row (or column) entry is zero for both the master and slave sets of integers, a loop over all rows (or columns) is implied, generating \( nrow \) (or \( ncol \)) separate constraints, where \( nrow \) and \( ncol \) are the number of rows and columns in the shell unit. For example, the G-2c record

\[
2 \ 0 \ 1 \ 3 \ 5 \ \ 2 \ 0 \ 1 \ 4 \ 3 \ 5
\]
means that for each row of nodal layer 5, the \( w \) freedom (\( \text{IDIR} = 3 \)) on column 14 is equated with the \( w \) freedom on column 1. When this capability is used with two different shell units, the number of rows (columns) must of course be the same.

If the value of \( \text{IR2} \) or \( \text{IC2} \) (for the slave nodes) is negative, the compatibility between units \( \text{IU2} \) and \( \text{IU1} \) is enforced in reverse order (this is identical to what happens with the \( \text{NBOUND} \) variable on G-1 records). This option is used if the shell coordinate \( X \) or \( Y \) in the two connected units is increasing in opposite directions along the common boundary.

Since no transformations are performed, the specified nodes should not lie on a boundary line involved in a shell unit connection (G-1) unless the two shell units have the same orientation with respect to the global coordinates.

As is the case with the G-2 record, a special option is provided for joining shell units using G-2c/ G-2d record pairs. If the \( \text{ID1} \) and \( \text{ID2} \) variables are both zero, then full compatibility is enforced between the nodes on \( \text{IU1} \) and \( \text{IU2} \). This means that the components on the node belonging to \( \text{IU2} \) are first transformed into the local system in \( \text{IU1} \) before the compatibility condition is enforced, which is the same thing that happens when shell units are joined using G-1 records. The zero rule also applies to the row and column indices here; thus, to join shell unit 2, row 1, nodal layer 5 to shell unit 1, row 13, nodal layer 7, the single record

\[
1 \ 13 \ 0 \ 0 \ 7 \ 2 \ 1 \ 0 \ 0 \ 5
\]

suffices. Here, row 13 could be in the interior of the unit. The user must be very careful when using this option. First, the user must insure the same row or column compatibility as required for the G-1 records; second, the user must insure that the lines involved in the juncture actually coincide in space, since the code will not check for the physical coincidence of the juncture lines. The user can take advantage of \( \text{IGLOBE} \) options 3, 4, or 5 (M-1) to move the units around until they meet in the right place.

The next six items (\( \text{ND1A} \) through \( \text{INC2} \)) on the G-2c record are optional input variables allowing the user to specify full compatibility for a range of nodes. These data are ignored unless the following logical expression evaluates to ‘True’:

\[
\{(\text{IR1} = 0 \ or \ \text{IC1} = 0) \ and \ (\text{IR2} = 0 \ or \ \text{IC2} = 0)\}
\]

These optional input variables provide a convenient (interior-looping) method for specifying compatibility between any portion of a row or column of one unit and a portion of a row or column of another unit. This type of input may also be used in element units provided the range of nodes has a fixed increment. For both shell and element units, increasing and decreasing ranges are allowed. If \( \text{INC1} = 0 \) or \( \text{INC2} = 0 \), an increment of +1 is used for increasing ranges and -1 for decreasing ranges.
Typically, one G-2c record will be used to define each of the $\text{NPATX}$ partial displacement compatibility conditions for the problem at hand. For some configurations, however, it is convenient to use the G-2c exterior-looping feature described below to make two or more specifications with a single G-2c record. The final item (NDEFS) on the G-2c record is an optional control parameter that, when omitted or set equal to any integer less than or equal to unity, causes the G-2c record to generate a single (set of) partial displacement compatibility specification(s) — or, when NDEFS > 1, causes the G-2c record (with information on the G-2d record following it) to generate NDEFS sets of partial compatibility specifications (see the example following the G-2d description, below).

If the exterior-looping feature is not used on the G-2d record immediately following a G-2c record, the G-2c record defines a single set of partial displacement compatibilities; if the exterior-looping feature is used, the G-2c/G-2d record pair defines NDEFS sets of conditions.

<table>
<thead>
<tr>
<th>IU1</th>
<th>IR1</th>
<th>IC1</th>
<th>IL1</th>
<th>ID1</th>
<th>IU2</th>
<th>IR2</th>
<th>IC2</th>
<th>IL2</th>
<th>ID2</th>
<th>ND1A</th>
<th>ND1B</th>
<th>INC1</th>
<th>ND2A</th>
<th>ND2B</th>
<th>INC2</th>
</tr>
</thead>
<tbody>
<tr>
<td>IU1</td>
<td>unit number of first node; $\text{IU1} = \text{NUNITS} + k$ indicates element unit $k$; $\text{IU1}$ must be the lower number ($\text{IU1} \leq \text{IU2}$); see CAUTION, below.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IR1</td>
<td>row number of first node or node number in element unit; if $\text{IR1} = 0$, then either $\text{IR2} = 0$ or $\text{IC2} = 0$ must be set.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IC1</td>
<td>column number of first node. $\text{IC1} = 0$ in element unit; if $\text{IC1} = 0$, then either $\text{IR2} = 0$ or $\text{IC2} = 0$ must be set.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IL1</td>
<td>nodal layer number for the first node; set $\text{IL1} = 0$ if unit $\text{IU1}$ is a single-layered shell unit, or to a positive value not exceeding the number of layers of nodes in unit $\text{IU1}$ if it is a multi-layered (solid) shell unit.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID1</td>
<td>Indicates the component involved at the first node; $\text{ID1} = 1, 2, 3, 4, 5$ or 6 for $u$, $v$, $w$, $ru$, $rv$, $rw$, respectively; if $\text{ID1} = 0$, then all components are involved, and $\text{ID2} = 0$ must be set.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IU2</td>
<td>unit number of second node; $\text{IU2} = \text{NUNITS} + k$ for element unit $k$; $\text{IU2}$ must be the higher number ($\text{IU2} \geq \text{IU1}$); see CAUTION, below.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IR2</td>
<td>row number of second node or node number in element unit.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IC2</td>
<td>column number of second node or node number in element unit.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IL2</td>
<td>nodal layer number for the second node; set $\text{IL2} = 0$ if unit $\text{IU2}$ is a single-layered shell unit, or to a positive value not exceeding the number of layers of nodes in unit $\text{IU2}$ if it is a multi-layered (solid) shell unit.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**ID2** indicates the component involved at the second node. **ID2** = 1, 2, 3, 4, 5 or 6 for \(u, v, w, ru, rv, rw\), respectively; if **ID2** = 0, then the components are first transformed to the \(IU1\) system before partial compatibility is enforced. Note: if **ID2** = 0, then **ID1** must also be zero.

**ND1A** first node of: (row, if \(IR1 = 0\)) or (column, if \(IC1 = 0\))

**ND1B** last node of: (row, if \(IR1 = 0\)) or (column, if \(IC1 = 0\))

**INC1** increment for unit \(IU1\) nodes from **ND1A** to **ND1B**

**ND2A** first node of: (row, if \(IR2 = 0\)) or (column, if \(IC2 = 0\))

**ND2B** last node of: (row, if \(IR2 = 0\)) or (column, if \(IC2 = 0\))

**INC2** increment for unit \(IU2\) nodes from **ND2A** to **ND2B**

*CAUTION*: when different shell or element units are involved, the lower-numbered unit must be given first. This rule also applies to node numbers within a specific unit; the lower-numbered node must be given first. In shell units, nodes are numbered in row-major order; *i.e.*, along row 1, starting at column 1; then along row 2, starting at column 1; *etc.*

\(\rightarrow\) go to G-2d
**G-2d Partial Compatibility Looping and Incrementations**

A single type G-2d record must be included immediately after each type G-2c record. The *external looping* parameter (NDEFS) and ten incrementation variables are specified here for use with information provided on the G-2c record.

<table>
<thead>
<tr>
<th>NDEFS</th>
<th>JU1</th>
<th>JR1</th>
<th>JC1</th>
<th>JL1</th>
<th>JD1</th>
<th>JU2</th>
<th>JR2</th>
<th>JC2</th>
<th>JL2</th>
<th>JD2</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of partial compatibility specifications generated <em>via</em> this G-2c/G-2d record pair; set to unity by STAGS if nonpositive.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation variable for use with IU1 (unit-specification) variable on the parent G-2c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation variable for use with IR1 (on G-2c)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation variable for use with IC1 (on G-2c)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation variable for use with IL1 (on G-2c)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation variable for use with ID1 (on G-2c)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation variable for use with IU2 (on G-2c)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation variable for use with IR2 (on G-2c)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation variable for use with IC2 (on G-2c)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation variable for use with IL2 (on G-2c)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation variable for use with ID2 (on G-2c)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero or positive.

**Example:** the following G-2c/G-2d record combination:

```
63 2 0 3 4 71 2 0 3 4 1 7 1 1 7 1 $G-2c record
3 0 0 0 0 1 0 0 0 0 1 1 7 1 1 7 1 $G-2d record
```

generates 3 sets of partial compatibility definitions, using the *interior-looping* feature of the G-2c record to enforce compatibility between nodes on row 2 of (nodal layer 3) of shell units 71 and 63, and using the *exterior-looping* feature of the G-2d record to specify *direction-5* and *direction-6* components in addition to the *direction-4* component that is specified on the G-2c record.
NPATX (B-2) number of G-2c type partial-compatibility records required
NCONST (B-2) number of Lagrange-constraint equations
NINSR (B-2) number of inserted node-element sets defining cracks
NUNITE (B-2) number of element units

if (NPATX G-2c records have been defined) then
  elseif (NCONST < 0) then  go to  G-3
  elseif (NINSR > 0) then  go to  G-5
  elseif (NUNITE > 0) then  go to  H-1
  else  follow instructions at end of  H-1
else  return to continue defining  G-2c
G-3 Constraint—Record 1

This record is read if NCONST < 0 (B-2). The sequence G-3, G-4 is repeated [NCONST] times, each sequence defining one or more Lagrange constraints of the type

\[ \sum_{i=1}^{n} c_{i}x_{i} + c_{0} + c_{A}P_{A} + c_{B}P_{B} = 0 \]

where

- \( x_{i} \) are displacement unknowns, scaled by constant coefficients, \( c_{i} \);
- \( c_{0} \) is a constant term;
- \( P_{A} \) and \( P_{B} \) are the load factors (see 6.5 “Loads” on page 6-64), scaled by constant coefficients, \( c_{A} \) and \( c_{B} \);

and where the number of constraints defined via any given G-3, G-4 record sequence depends on whether or not the exterior looping feature, described subsequently, is used.

In general, Lagrange constraints have \( n + 3 \) terms, where \( n \geq 1 \) is required. Each of the remaining three terms is optional. The displacement unknowns \( (x_{i}) \) are defined by specification of unit number, local node number, and dof direction. The constant term \( (c_{0}) \) and the constant coefficients \( (c_{i}, c_{A}, c_{B}) \) are defined as real numbers.

Each Lagrange constraint is enforced by one of two methods, described below, which is independently specified for each constraint. The direct-elimination method is generally preferable, as each constraint of this type removes an equation, whereas selection of the constraint-equation method adds an equation.

**Constraint-equation method**

Constraint equations are assembled into the stiffness matrix along with the other freedoms. For each constraint, a new freedom is introduced. In order to avoid numerical difficulties, it may be necessary to scale constraint equations. This is achieved by multiplying the constant coefficients \( (c_{i}, c_{A}, c_{B}) \), as well as the constant term \( (c_{0}) \), by a scale factor, which should be chosen so that each term is roughly the order of magnitude of the other stiffness matrix elements. We suggest a scale factor equal to the material modulus or modulus times wall thickness.
One additional feature of Lagrange constraints is that for each equation a negative root will appear in the stiffness matrix. It is important to take this into account when using properties of the stiffness matrix to determine structural stability characteristics.

Choose the constraint-equation method when it is difficult to select a set of unique dependent freedoms from the total set of freedoms involved in all constraints; see NOTE, p. 5-43. Constraint equations are very general, and there are no limitations whatsoever regarding ordering of terms in a constraint equation or on the appearance of \( \text{dof} \) in other constraints, including direct elimination constraints.

**Direct-elimination method**

This results in one freedom, the *dependent freedom*, being defined in terms of the other freedoms, the *independent freedoms*, and eliminated from the system of equations. Thus, rather than adding a constraint equation, this method results in the elimination of one equation for each constraint. Scaling, as in the constraint-equation method, is not necessary for direct elimination.

Choose the direct-elimination method when it is possible to express a dependent freedom as a linear combination of “independent” freedoms. By independent, we mean that the particular freedom does not appear as a dependent freedom in any other direct-elimination constraint.

A maximum of 100 terms in a single constraint is allowed. Given the very poor bandwidth such a large number of terms creates, this is a very generous limit. A very effective way around the limitation on the number of terms and the resulting bandwidth problems is to nest constraints. Nesting means to define intermediate nodes first and break up the constraints into smaller components. For example, if freedom \( k \) equals the sum of 400 particular freedoms, one could define 8 extra freedoms. On the lowest level, 8 freedoms are each set equal to the sum of 50 different freedoms taken from the set of 400, forming the inner level of the nest. The outer level is simply a constraint setting the desired freedom to be the sum of the 8 newly-defined freedoms. Nesting can, in principal, be carried to any level. The bandwidth is limited to the maximum number of terms in a given constraint.
NTERMS  NX  INC1  INC2  INC3  INC4  INC5

NTERMS  \[|\text{NTERMS}|\] defines the number of terms in the constraint; \(|\text{NTERMS}| \leq 100\).
NTERMS > 0  specifies the constraint-equation method.
NTERMS < 0  specifies the direct-elimination method.

NX  Looping parameter; defines the number of Lagrange constraints to be generated with information on the current G-3, G-4 record sequence; set equal to unity by STAGS if nonpositive or omitted.

INC1  Incrementation parameter for all of the IU variables specified on G-4 records in the current G-3, G-4 sequence (used only if NX > 1).

INC2  Incrementation parameter for all of the IX variables specified on G-4 records in the current G-3, G-4 sequence (used only if NX > 1).

INC3  Incrementation parameter for all of the IY variables specified on G-4 records in the current G-3, G-4 sequence (used only if NX > 1).

INC4  Incrementation parameter for all of the ID variables specified on G-4 records in the current G-3, G-4 sequence (used only if NX > 1).

INC5  Incrementation parameter for all of the IZ variables specified on G-4 records in the current G-3, G-4 sequence (used only if NX > 1).

EXAMPLE:  \[
\text{NTERMS } = 5 \] specifies one or more constraints with 5 terms, implemented via the constraint-equation method.
\[
\text{NTERMS } = -5 \] specifies one or more constraints with 5 terms, implemented via the direct-elimination method.

\[ \text{go to G-4} \]
G-4 Constraint—Record 2

This record is read \texttt{NTERMS} times in each G-3/G-4 sequence. The number of Shell Units is \texttt{NUNITS} (B-2). For element unit \textit{l}, the variable \texttt{IU} is set to \textit{k}, where \textit{k = NUNITS + l}. Constraints on displacement or rotation components at a juncture line must be defined in terms of the freedoms on the shell unit with the lowest number of those involved in the juncture. Constraints at element unit node points located at shell unit nodes must be defined in terms of the freedoms at the shell unit node.

\textit{CAUTION}: Slave \textit{dof} may not appear in constraint equations. Master/slave \textit{dof} relationships are created automatically by \textsc{stags} where compatibility has been prescribed. There are three different modeling features which result in such relationships:

- shell-unit junctures (G-1)
- displacement compatibility (G-2)
- nodal definition in a \textit{higher-numbered} unit by reference to a \textit{lower-numbered} unit (S-1)

\begin{tabular}{|c|c|c|c|c|}
\hline
\texttt{IU(i)} & \texttt{IX(i)} & \texttt{IY(i)} & \texttt{ID(i)} & \texttt{CC(i)} & \texttt{IZ(i)} \\
\hline
\end{tabular}

\texttt{IU(i)} \texttt{= 0} indicates that either the constant term (\textit{c0}) or one of the load-factor coefficients (\textit{cA}, \textit{cB}) is defined. Currently, only one record defining the constant term (\texttt{IU(i)=0}) is allowed (per constraint), and this record must be the last one defined for this constraint.

\texttt{IU(i) > 0} indicates that one of the \textit{dof} coefficients (\textit{ci}) is defined; \texttt{IU(i)} specifies the unit where the unknown \textit{x} is located.

\begin{align*}
\texttt{IU(i) = IUNIT} & \quad \text{for a shell unit, where IUNIT is the unit number} \\
\texttt{IU(i) = k} & \quad \text{for element unit \textit{l} (see above)}
\end{align*}

\textit{NOTE}: In a direct-elimination constraint, the first freedom specified is taken to be the dependent freedom, the one that is eliminated. A dependent freedom must not otherwise be constrained by a BC; and it must not appear in any other direct-elimination constraint, either as the dependent freedom or as an independent freedom. It may appear in any number of constraint equations.

\texttt{IX(i)} node/row where the unknown \textit{x} is located. \texttt{IX(i)} is defined as:

\begin{align*}
\texttt{IU(i) = IUNIT: IX(i) = row number} \\
\texttt{IU(i) = k: IX(i) = node number}
\end{align*}
\( IY(i) \)

Column where the unknown \( x_i \) is located. \( IY(i) \) is defined as:

\[
IU(i) = IUNIT: \quad IY(i) = \text{column number}
\]

\[
IU(i) = k: \quad IY(i) = \text{irrelevant}
\]

\( ID(i) \)

Term identifier, interpreted according to the value of \( IU(i) \), as follows:

\[
IU(i) = 0
\]

1. Load-factor coefficient \( c_A \)
2. Load-factor coefficient \( c_B \)
3. Constant term, \( c_0 \)

\[
IU(i) > 0
\]

1. \( dof \) coefficient; \( x_i \) refers to the displacement component \( u \)
2. \( dof \) coefficient; \( x_i \) refers to the displacement component \( v \)
3. \( dof \) coefficient; \( x_i \) refers to the displacement component \( w \)
4. \( dof \) coefficient; \( x_i \) refers to the displacement component \( ru \)
5. \( dof \) coefficient; \( x_i \) refers to the displacement component \( rv \)
6. \( dof \) coefficient; \( x_i \) refers to the displacement component \( rw \)

\( CC(i) \)

Coefficient as required by the specification of \( ID(i) \), above.

\( IZ(i) \)

Nodal layer (in a shell unit) to which this constraint applies.

**Example:** the following G-3/G-4 record sequence:

\[
\begin{align*}
2 & 3 0 0 0 1 \\
1 & 511 0 4 1.000E6 1 \\
1 & 515 0 4 -1.000E6 1 \\
\end{align*}
\]

generates the same three Lagrange constraint definitions as the following three G-3/G-4 record sequences:

\[
\begin{align*}
2 & \text{ G-3 record for definition \# 1} \\
1 & 511 0 4 1.000E6 1 \text{ G-4 record for definition \# 1} \\
1 & 515 0 4 -1.000E6 1 \text{ G-4 record for definition \# 1} \\
2 & \text{ G-3 record for definition \# 2} \\
1 & 511 0 5 1.000E6 1 \text{ G-4 record for definition \# 2} \\
1 & 515 0 5 -1.000E6 1 \text{ G-4 record for definition \# 2} \\
2 & \text{ G-3 record for definition \# 3} \\
1 & 511 0 6 1.000E6 1 \text{ G-4 record for definition \# 3} \\
1 & 515 0 6 -1.000E6 1 \text{ G-4 record for definition \# 3} \\
\end{align*}
\]
Only one of the five incrementation parameters should be used with any given G-3/G-4 record sequence.

- **NTERMS (G-3)**: number of terms in constraint equation
- **NCONST (B-2)**: number of Lagrange-constraint-definition sequences
- **NINSR (B-2)**: number of inserted node/element sets defining cracks
- **NUNITE (B-2)**: number of element units

if (NTERMS G-4 records have been defined) then
  if (NCONST G-3 records have been defined) then
    if (NINSR > 0) then go to G-5
    elseif (NUNITE > 0) then go to H-1
    else follow instructions at end of H-1
  else return to G-3
else continue defining G-4
G-5 Crack Inserted Node Set—Record 1

G-5 is included when \textbf{NINSR} > 0 \textbf{(B-2)}. The G-5–G-7 series is repeated \textbf{NINSR} times.

Generation of a crack requires the association of an ordered set of nodes in the structural model with the crack face; these nodes must be ordered either from left end of the crack to the right end of the crack, or \textit{vice versa}. These nodes are collocated with nodes already defined in the model but have independent displacement components after the crack opens. A number of elements must also be specified to show which elements are to use the new nodes instead of the original nodes. All crack nodes required for a series of analyses must be defined before the first execution. The growth of a crack during a static or dynamic analysis is defined by choosing a subset of such nodes to be “open,” with the freedoms on the remainder of the nodes (closed nodes) set (or “slaved”) to the corresponding node on the other side of the crack. As the crack grows, a larger number of nodes is designated as open. In default operation, STAGS keeps a record of the growth of the crack, so that upon restart, the crack history at the desired restart step is retrieved and used. A solution option exists to override the STAGS history at restart and use a new crack configuration specified in these records.

If the user has already generated slave nodes corresponding to a crack, then he can specify these nodes on record G-6. In this case, G-6 consists of \textit{two} sets of crack nodes. The first set refers to the master nodes, and the second to the slave nodes. The lists must be of identical length so that a one-to-one correspondence between the first list and second list exists. No G-7 records are needed when the user has already defined the slave nodes and the elements that contain them.

If the user inputs only one list on G-6, STAGS generates the second set of crack nodes. In this case, if a section of a crack lies along a line that contains two sets of nodes (slave/master relation defined by G-1 or G-2 records), the crack nodes in the G-6 record should refer only to the slave nodes. The G-7 records should refer only to elements which contain these slave nodes.

More than one crack can be defined in a given model, with each crack given a unique identifier \textbf{(NCRACK)} which starts with one and increments by one for every new crack defined. If \textbf{NCRACK} < 0, a hinged crack is created with the identifier \textbf{|NCRACK|} (see the parameter definition list).

STAGS can be made to grow a crack automatically along the predetermined path defined by these records, given a crack growth criterion. At present, only the Crack Tip Opening Angle (CTOA) criterion is implemented (see \cite{1}, \textit{for example}). If the variable \textbf{ITEAR} is nonzero, the variable \textbf{ACRIT} is interpreted as the CTOA that must be attained before the crack tip node is automatically released. STAGS will continue the execution with the strategy chosen by the user until some other termination criteria are reached, or the crack tip exhausts the list of closed nodes defined on the given crack. More than one crack with CTOA growth can be defined.

* Numbers in square brackets, here, indicate references at the end of this chapter.
Discretization and Connectivity Summary Model Input

NCRACK INNODS INELTS ITEAR ACRIT CTOD
ACRITT SAWL SAWT CSSCALE IDREC

NCRACK  crack identifier; if NCRACK < 0, only the translations will be constrained along closed portions of a crack; this is the so-called “hinged crack”

INNODS  number of inserted node records (G-6)

INELTS  number of element records (G-7)

ITEAR  automatic crack growth indicator

  0  – no automatic crack extension; ACRIT is ignored
  1  – crack extension according to the CTOA criterion
-1  – crack extension according to CTOA, but along a symmetry boundary (where only the upper or lower surface of the crack is actually modeled)

ACRIT  the critical opening angle (CTOA), in degrees, for the L (longitudinal) direction; always input the full angle, even when the crack is along a symmetry boundary (ITEAR = -1)

CTOD  the distance from the crack tip to the point from which ACRIT is measured; if CTOD is 0, the node adjacent to the tip is used; if ACRIT cannot be generated with the user’s CTOD, the adjacent node is also used

ACRITT  the critical opening angle (CTOA), in degrees, for the T (transverse) direction, if different from ACRIT; if zero, ACRITT is set to ACRIT inside STAGS

SAWL  the critical opening angle (CTOA), in degrees, or the longitudinal direction for the first node released in a sawcut; if zero, STAGS uses ACRIT

SAWT  the critical opening angle (CTOA), in degrees, for the transverse direction for the first node released in a sawcut; if zero, STAGS uses SAWL

CSSCALE  scale factor for Lagrangian constraints used to close the crack; recommended value is 1.0 (the default value, if not specified)

IDREC  component index for the shell normal coordinate; defaults to 3 if zero or omitted

go to G-6
G-6 Crack Inserted Node Set—Record 2

The nodes defining a crack are specified here. G-6 is repeated INNODS (G-5) times. The ICUNIT, JCUNIT and ICOPEN parameters on this record have special roles, as described in the following paragraphs.

If the unit that is identified by the ICUNIT parameter has sandwich or solid elements, the sign of ICUNIT specifies that the crack is on the lower or on the upper surface of that unit. In particular, the crack will be on the lower surface of unit ICUNIT if $ICUNIT > 0$, or it will be on the upper surface of unit $|ICUNIT|$ if $ICUNIT < 0$.

If JCUNIT is greater than zero, the set JCRW1, JCCOL1, JCRW2, and JCCOL2 refer to a second set of nodes already defined in STAGS that are associated one-to-one with the first set specified by ICUNIT, ICROW1, ICCOL1, ICROW2, and ICCOL2. When the crack nodes are open, the corresponding associated node on the other side of the crack is independent; when closed, it is slaved. If JCUNIT is zero, STAGS generates the other set of crack nodes; and a G-7 record specifying the elements belonging to the new nodes must be provided.

ICOPEN requires special treatment when there is a line of symmetry (ITEAR = -1). When there is no line of symmetry, any positive value of ICOPEN will cause the released node to be completely free. However, this procedure is not always suitable for symmetry, since two symmetry lines can cross at a corner. For this special case, we have added two additional options, as listed below. In addition, if the user defines his own slave crack nodes (JCRW1, JCCOL1, JCRW2, JCCOL2), the released nodes always have the user-defined freedom pattern (see S-1, for example).
ICOPEN  ICUNIT  ICROW1  ICCOL1  ICROW2  ICCOL2  JCUNIT  JCROW1  JCCOL1  JCROW2  JCCOL2  THETA

ICOPEN

0 – node is “closed” or “slaved” to the parent node on the other side of the crack

1 – node is “open” or independent, with its own freedom set. Use this choice when ITEAR = 0 (G-5 – no line of symmetry). New node is completely free (full set of freedoms).

1 – if ITEAR = -1 (symmetry) input 1 if the symmetry line is along a row in a shell unit. For an element unit, release u and Rv.

2 – if ITEAR = -1 (symmetry) input 2 if the symmetry line is along a column in a shell unit. For an element unit, release v and Ru.

3 – treat the adjacent tip as a sawcut. The CTOA for the first node refers to SAWL or SAWT; newly released nodes default back to the original ACRIT or ACRITT. In all other respects, this option is identical to ICOPEN = 1.

4 – treat the adjacent tip as a sawcut. The CTOA for the first node refers to SAWL or SAWT; newly released nodes default back to the original ACRIT or ACRITT. In all other respects, this option is identical to ICOPEN = 2.

5 – ignore all symmetry and free up the node unconditionally. For ITEAR = 0, all positive choices for ICOPEN are equivalent after the first node has opened.

For conditions not covered by the above options, the user can arbitrarily define his own slave node set boundary conditions (see discussion above). Another way to override the automatic freeing of dependent crack nodes is to assign them a vanishingly small specified displacement, which causes the opened node to inherit that constraint.

ICUNIT

unit number; a given crack can span more than one unit. Remember that crack nodes must be defined in order along the crack face, even when spanning more than one unit (this requires more than one G-6 record).

ICROW1

row number of first node (shell unit); number of first node (element unit)

ICCOL1

column number of first node (shell unit); number of last node (element unit)

ICROW2

row number of last node (shell unit) Increment in passing from node ICROW1 to ICCOL1 (element unit); nodes must be numbered in order along the crack face.

ICCOL2

column number of last node (shell unit) (not used in element unit); nodes must be numbered in order along the crack face.

JCUNIT

unit number containing nodes already defined that will form the other side of the crack.
JCROW1  row number of first node (shell unit); number of first node (element unit).
JCCOL1  column number of first node (shell unit); number of last node (element unit).
JCROW2  row number of last node (shell unit); increment in passing from node JCROW1 to JCCOL1 (element unit); nodes must be numbered in order along the crack face and correspond one-to-one with nodes on the other side of the crack.
JCCOL2  column number of last node (shell unit) (not used in element unit); nodes must be numbered in order along the crack face and correspond one-to-one with nodes on the other side of the crack.
THETA  used only when ACRITT is nonzero (G-5), or is different from ACRIT. THETA is equal to the angle (in degrees) between the longitudinal direction associated with ACRIT and the x computational coordinate direction (x’).

INNODS (G-5)  number of node records

if (INNODS G-6 records have been defined) then  go to  G-7
else  continue defining  G-6
G-7 Crack Inserted Node Set—Record 3

In most situations, the elements associated with inserted crack nodes can be defined either on one side of the crack or the other. Remember to include the elements associated with the crack tip nodes. G-7 is repeated INELTS (G-5) times.

Note: If a beam on a shell unit is specified (JETYPE=1), row and column identifiers must be provided for both ends of the beam. Hence, either JCRW2 must be different from JCRW1, or JCCOL2 must be different from JCCOL1. This requirement eliminates the ambiguity between a beam that lies parallel to the crack and one that is perpendicular to the crack.

<table>
<thead>
<tr>
<th>JETYPE</th>
<th>JCUNIT</th>
<th>JCRW1</th>
<th>JCCOL1</th>
<th>JCRW2</th>
<th>JCCOL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

JETYPE 0 – skin elements

JETYPE 1 – stiffener elements

JCUNIT unit in which elements defined below are referenced.

JCRW1 row number of first node (shell unit); number of first element (element unit)

JCCOL1 column number of first node (shell unit); number of last element (element unit)

JCRW2 row number of last node (shell unit); increment in passing from element JCRW1 to JCCOL1 (element unit); elements can appear in any order.

JCCOL2 column number of last node (shell unit) (not used in element unit); elements can appear in any order.

Example:

To illustrate this feature, let us assume that a STAGS model with an element edge length of 0.0625 inches has been created and that a crack is to be inserted within shell unit number 5. Let us also assume that a portion of the crack is defined as being initially “open” and the remainder is defined as “closed.” The self-similar crack growth path for this crack is specified by all of this. The input records are

1 2 1 1 4.6 0.0625 $ G-5 Crack-inserted node set
1 5 5 1 5 32 $ G-6 Open crack (2 inches long)
0 5 5 33 5 97 $ G-6 Closed crack (4 inches long)
0 5 4 1 4 96 $ G-7

The G-5 record indicates that only one crack is defined, that two inserted node records are used, that automatic crack extension by the CTOA criteria will be used, that the critical crack tip
opening angle (CTOA) is 4.6 degrees, and that the point where the opening angle is calculated
is 0.0625 inches from the crack tip. The first G-6 record indicates that this crack is “open”, that
it is in unit 5, and that it is open from row 5 and column 1 up to row 5 and column 32. This is
equivalent, assuming uniform grid spacing of 0.0625 inches along the crack, to a 2-inch-long
initial crack (32 elements with crack tip at row 5 and column 33). The second G-6 record
indicates that this crack segment is “closed” initially but potentially can “open” during the
analysis, that the crack is in unit 5, that the crack tip is at row 5 and column 33, and that it could
grow along row 5 up to column 97 (i.e., that there is a 4-inch-long potential crack growth). The
G-7 record indicates that crack is in skin elements, that unit 5 contains the crack, that the crack
definition starts in the fourth row and first column of elements, and it can potentially grow to the
96th column of elements in the fourth row.

\[
\begin{align*}
\text{INELTS (G-5)} & \quad \text{number of element records} \\
\text{NINSR (B-2)} & \quad \text{number of inserted node/element sets defining cracks} \\
\text{NUNITE (B-2)} & \quad \text{number of element units} \\
\end{align*}
\]

if \((\text{INELTS G-7 records have been defined})\) then
  if \((\text{NINSR inserted node/element sets have been defined})\) then
    if \((\text{NUNITE} > 0)\) then go to H-1
    else follow instructions at end of H-1
  else return to G-5
else continue defining G-7
H-1 Element Unit Summary

If NUNITE > 0 on the user’s B-2 record, then a single H-1 record is required for each of the NUNITE element units in the model. The H-1 record for a given element unit specifies (or postpones the specification of) one parameter (NUPT) that is used to select the method to be used in the node-specification process and six parameters (NT1, NT2, NT3, NT4, NT5 and NS5) that are used to select the method(s) and element(s) to be used in the element-specification process for that element unit, as described below. H-1 also allows the user to specify three parameters (IUWP, IUWE and IUDIMP) that trigger the employment of user-written subroutines to define some or all of the nodes, elements and/or imperfections in that element unit. H-1 also allows the user to specify a single parameter (IUWLE) that tells STAGS that user-defined element numbers may (or may not) be employed in that element unit.

For each element unit, the user must identify and define the location of each node point that is referenced in that element unit. This may include a subset of some or all of the nodes that are defined in one or more shell units and/or a set of auxiliary nodes that are defined in the element unit itself with their global system coordinates. In any event, the complete set of nodes that are involved in the element unit is referred to as its user points. The current version of STAGS gives the user two protocols for using “regular” data records to define some or all of these user points. These protocols are described in Chapters 8 and 9 of this document. STAGS also supports the employment of user-written subroutines to define some or all of the user points in the element unit. This approach is described in Chapter 12. Alternatively, PATRAN may be used to model the structure and output a STAGS input file (see Appendix D – PAT2STAGS).

The current version of STAGS gives the user two protocols for using “regular” data records to define the elements that are used in a given element unit. The first of these—the Edef (element–definition) protocol that has been used in every version of STAGS since finite elements and element units were introduced—is described in Chapter 8; and the second—the Ecom (element–command) protocol that uses a command-line approach and gives the analyst much more control and flexibility in the element–specification process—is described in Chapter 9. The current version of STAGS accommodates the use of either or both of these protocols—as the user wishes and/or as convenience dictates. STAGS also supports the employment of user-written subroutines to define some or all of the elements in the element unit. See Chapter 12 for more information about this.

The current version of STAGS gives the user single protocols for employing “regular” data records to define point and distributed loadings on the nodes and elements of the element unit.
These are described in Chapter 10 of this document. **STAGS** also supports the employment of *user-written subroutines* to define some or all of these loadings. With the current version of **STAGS**, imperfections in an element unit can only be accommodated *via* user-written subroutines. See Chapter 12 for more information about this.

**NUNITE** records of the following type are required:

<table>
<thead>
<tr>
<th>NUPT</th>
<th>NT1</th>
<th>NT2</th>
<th>NT3</th>
<th>NT4</th>
<th>NT5</th>
<th>IUWP</th>
<th>IUWE</th>
<th>IUDIMP</th>
<th>IUWLE</th>
<th>NS5</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUPT</td>
<td>control parameter for defining user points</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NT1</td>
<td>control parameter for defining “spring” elements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NT2</td>
<td>control parameter for defining “beam” elements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NT3</td>
<td>control parameter for defining “triangular” elements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NT4</td>
<td>control parameter for defining “quadrilateral” elements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NT5</td>
<td>control parameter for defining “other” elements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IUWP</td>
<td><em>user-points-via-subroutines</em> parameter:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 – all user points are defined on regular data records</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 – some user points are defined in user-written USRPT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IUWE</td>
<td><em>elements-via-subroutines</em> parameter:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 – all elements are defined on regular data records</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 – some elements are defined in user-written USRELT ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IUDIMP</td>
<td><em>imperfections-via-subroutines</em> parameter:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 – no initial imperfections</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 – initial imperfections are defined in user-written DIMP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IUWLE</td>
<td><em>user-element-numbers</em> parameter:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 – no user element numbers are specified</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 – user element numbers are specified (in element units)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NS5</td>
<td><em>line–line-contact-definitions</em> parameter:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 – no line-contact-interaction specifications</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&gt;0 – number of line-contact-interaction specifications</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note 1:** **STAGS** gives the user two methods for specifying the user points that are utilized in an element unit: the so-called *upts* protocol that has been used in every version of **STAGS** since element units were introduced, and the *user-points* protocol that gives the analyst more flexibility in the point-specification process. The older *upts* protocol is supported for use with
existing models and for use with any new models that may be constructed manually or with software that does not “know” about and/or support the newer user-points protocol. The value that the user specifies for the NUPTS parameter, on H-1, determines which of these two protocols is to be employed:

**NUPTS** = 0 — the user-points protocol is to be employed, to define an unspecified number of user points (where the user lets STAGS count the points)

**NUPTS** > 0 — the upts protocol is to be employed, to define NUPTS user points

**NUPTS** = -1 — no user points are to be defined

The point-definition protocols are described fully in Chapter 7 of this document.

**Note 2:** STAGS gives the user two methods for specifying the elements that are utilized in an element unit: the older (and more rigid) Edef (element-definition) protocol that has been used in every version of STAGS since finite elements and element units were introduced, and the newer (and more flexible) Ecom (element-command) protocol that uses a command-line approach. The older of these—the Edef protocol—is supported for use with existing models and for use with any new models that may be constructed manually or with software that does not “know” about and/or support the new Ecom protocol. As noted above, the current version of STAGS allows the user to employ either or both of these protocols—as the user wishes and/or as convenience dictates. The choice of which of these two methods is to be used (or to use both methods) is made via the values that are specified for the NT1, NT2, NT3, NT4 and NT5 parameters on H-1:

If one or more of these parameters is positive, STAGS starts its element-specification operations by using the historic Edef protocol (which is described in Chapter 8) to define NT1 “spring” (type E110, E120, E121 and/or E130) elements, NT2 “beam” (type E210 and/or E250) elements, NT3 “triangle” (type E320 and/or E330) elements and/or NT4 “quadrilateral” (type E410, E411, E420, E430, E480, E510 and/or E710) elements. After doing that, STAGS examines the NT5 parameter to determine what to do next. If NT5 = 0 on H-1, STAGS stops processing element-specification records and starts processing loading specifications. If NT5 = 1 on H-1, STAGS continues its element-specification operations by reading a type T-5 control record (see Chapter 8) and using the parameters that are specified there to define one or more type (type E810, E820 and/or E822 (contact) elements, one or more type E830, E840, E845, E847 and/or E849 (sandwich) elements and/or one or more of the E880-family elements—and then moves forward to the processing of loading specifications. If NT5 = 2 on H-1, STAGS continues its element-specification operations by reading a type T-5 control record
(Chapter 8) and using the parameters specified there to define one or more contact, sandwich and/or solid elements—and then switches over to the Ecom protocol to define one or more elements of any type (as described in Chapter 9), before moving forward to the processing of loading-specification records.

If the NT1, NT2, NT3, NT4 and NT5 parameters are all zero on H-1, STAGS recognizes the fact that the user only needs or wishes to employ the newer Ecom protocol for all element-specification operations. See Chapter 9 for more information about this protocol.

If the user wishes to construct an element unit that has user points but does not have any elements, he (or she) should set NT1, NT2, NT3, NT4 or NT5 equal to -1 on H-1.

Note 3: In no case does the inclusion of user-written element-definition subroutines preclude transmission of additional information on regular or on Ecom data records.

NUNITE (B-2) number of element units
NTAM (B-3) number of entries in the Material Table
NTAMT (B-3) number of entries in the Mount Element Table
NGCP (B-3) GCP-utilization flag
NTAB (B-3) number of entries in the Cross Section Table
NTAW (B-3) number of entries in the Wall Fabrication Table
NTAP (B-3) user parameters flag
NUNITS (B-2) number of shell units

if (NUNITE H-1 records have been defined) then
  if (NTAM > 0) then go to I-1
  elseif (NTAMT > 0) then go to I-4a
  elseif (NGCP > 0) then go to I-5a
  elseif (NTAB > 0) then go to J-1
  elseif (NTAW > 0) then go to K-1
  elseif (NTAP > 0) then go to L-1
  elseif (NUNITS > 0) then go to M-1
  else follow instructions at the end of R-3
else continue defining H-1
5.6 Data Tables

The records in this group (I–L) contain information as summarized below:

- I records: Material Table & Mount Element Table; GCP Materials & GCP Fabrications Tables
- J records: Cross Section Table
- K records: Wall Fabrication Table
- L records: User Parameters
I-1 Material Properties

Records I-1 and I-2 must be defined for each Material Table material. Furthermore, I-3 must be defined where plastic properties are specified, and I-3a must be defined where creep properties are specified. Creep properties cannot be defined for an elastic material.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITAM</td>
<td>user-assigned material number; $1 \leq \text{ITAM} \leq \text{NTAM}$ (B-3)</td>
</tr>
<tr>
<td>NESP</td>
<td>number of points on the $\varepsilon - \sigma$ curve (I-3), excluding the origin, which is defined to be $(\varepsilon, \sigma) = (0, 0)$. If plasticity is not included for this material, set $\text{NESP} = 0$. $\text{NESP}$ cannot be greater than 10.</td>
</tr>
<tr>
<td>IPLST</td>
<td>plasticity theory</td>
</tr>
<tr>
<td></td>
<td>0, 1 – White-Besseling theory</td>
</tr>
<tr>
<td></td>
<td>2 – isotropic strain hardening — INACTIVE</td>
</tr>
<tr>
<td></td>
<td>3 – kinematic strain hardening — INACTIVE</td>
</tr>
<tr>
<td></td>
<td>4 – deformation theory — INACTIVE</td>
</tr>
<tr>
<td>ITANST</td>
<td>material stiffness approximation</td>
</tr>
<tr>
<td></td>
<td>0 – initial strain</td>
</tr>
<tr>
<td></td>
<td>1 – tangential stiffness</td>
</tr>
<tr>
<td>ICREEP</td>
<td>creep flag — INACTIVE</td>
</tr>
<tr>
<td></td>
<td>0 – no creep</td>
</tr>
<tr>
<td></td>
<td>1 – creep</td>
</tr>
<tr>
<td>IPLANE</td>
<td>plane strain flag (White-Besseling plasticity or elastic only)</td>
</tr>
<tr>
<td></td>
<td>0 – plane stress theory</td>
</tr>
<tr>
<td></td>
<td>1 – plane strain theory</td>
</tr>
</tbody>
</table>

$\text{ITAM NESP IPLST ITANST ICREEP IPLANE}$

$\text{go to I-2}$
I-2 Material Elastic Properties

In plate or shell analysis the plane stress assumption is used; the transverse normal stress is assumed to be zero. In that case, the elastic properties are defined by a matrix $C$ such that

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \tau \end{bmatrix} = \begin{bmatrix} C \end{bmatrix} \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \gamma \end{bmatrix}$$

If the material is defined in terms of the principle directions (a normal stress in the principle directions causes no shear deformation), the material is orthotropic and

$$\begin{bmatrix} C \end{bmatrix} = \begin{bmatrix} E_1 & \frac{\nu_{12} E_1}{1 - \nu_{12} \nu_{21}} & 0 \\ \frac{\nu_{12} E_2}{1 - \nu_{12} \nu_{21}} & E_2 & 0 \\ 0 & 0 & G_{12} \end{bmatrix}$$

where $\nu_{12} E_1 = \nu_{21} E_2$. Materials must be defined with respect to the principle material directions ($\phi_1, \phi_2$) (i.e., $C_{13}, C_{23} = 0$).

If the material is isotropic, it is not necessary to input all the four material constants. When the shear modulus is set to zero, it is assumed that the material is isotropic, i.e., that $E_2 = E_1$ and $G = \frac{E_1}{2(1 + \nu_{12})}$. If a material with a zero shear modulus must be defined, input a very small value for $G$. If the coefficient of thermal expansion in the $\phi_2$ direction is set equal to zero, it is assumed that $\alpha_1 = \alpha_2$ ($A_1 = A_2$).
E1  U12  G  RHO  A1  E2  A2

E1  elastic modulus in $\phi_1$ direction

U12  Poisson’s ratio, $\nu_{12}$, where $\nu_{12}E_1 = \nu_{21}E_2$.

CAUTION: The $\nu$-subscript conventions used above are different from those used in many references. Be sure that the $\nu_{12}$ defined here satisfies the above relationship.

G  shear modulus, $G$. Setting $G = 0$ indicates an isotropic material; this causes enforcement of the $E_2 = E_1 = E$ and the $G = E/(2(1 + \nu))$ relationships; set $G$ to a small value to define a material with a zero shear modulus.

RHO  weight density, in units of force/volume (e.g., N/m$^3$ or lb/in$^3$)

A1  coefficient of thermal expansion in $\phi_1$ direction, $\alpha_1$

E2  elastic modulus in $\phi_2$ direction; input zero for isotropic materials

A2  coefficient of thermal expansion in $\phi_2$ direction, $\alpha_2$; setting $A2 = 0$ causes enforcement of the relationship $\alpha_2 = \alpha_1 = \alpha$.

NESP (I-1)  number of points on $(\sigma, \varepsilon)$ curve

if (NESP > 0)  then  go to I-3

else  follow instructions at end of I-3a
I-3 Material Plastic Properties

This record is included only if $\text{NESP} > 0$ (I-1). The two plasticity theories of choice available in STAGS are the White-Besseling theory (or mechanical sublayer model), and isotropic strain hardening. Both have their limitations but perform adequately for a moderate-strain analysis of metals. Each theory is based on a piecewise linear stress-strain curve, described on the data records by definition of strain and stress at the corners. The slope of the stress-strain curve must decrease at each corner. For a general layered shell, Simpson’s rule is used for integration through the thickness of each layer, while for a stiffener, the plastic strain is considered constant within each of a number of subelements. If, during a run, the strain exceeds the value of the last strain point given below, the code assumes that another segment beyond those input exists. This segment has a zero slope (perfectly plastic behavior), as shown in Figure 5.1. The first point must agree with elastic behavior, or the modulus times the strain; if this condition is not satisfied, a diagnostic is provided.

![Stress-strain curve](image)

**Figure 5.1** Stress-strain curve.

Definition of material plastic properties does not automatically include nonlinear material effects in an analysis. Parameters for each shell unit, stiffener, or element control whether or not plasticity is included for that structural segment. Users often find it convenient to specify plastic properties when defining a material and then selectively control plasticity effects as appropriate for each geometric region independently of the material definition records. Where plasticity has been suppressed, material behavior is elastic (I-2).
\((E(i), S(i)), i=1,NESP\)

**E(i)**  
\(\varepsilon_i\), the strain value corresponding to the \(i^{th}\) point on the \((\varepsilon, \sigma)\) curve. Note that the origin, \((\varepsilon, \sigma) = (0, 0)\), is *not* included as a data point. Additionally, \((E(1), S(1))\) must be defined to satisfy the relation \(\sigma_1 = E \varepsilon_1\); where \(\varepsilon_1 = E(1), \sigma_1 = S(1)\), and \(E = E1\) (the modulus of elasticity on l-2).

**S(i)**  
\(\sigma_i\), the stress value corresponding to the \(i^{th}\) point on the \((\varepsilon, \sigma)\) curve

**ICREEP** (l-1)  
creep properties flag

if \((ICREEP > 0)\) then  
*go to* l-3a
else  
*follow instructions at end of* l-3a
I-3a Material Creep Properties

INACTIVE

I-3a is included only if ICREEP > 0 (I-1). A simple power-law creep theory is implemented, with the effective creep strain $\dot{\varepsilon}^c$ dependent on creep time $t$ by

$$\dot{\varepsilon}^c = A (\bar{\sigma}/B)^m t^n,$$

where $\bar{\sigma}$ is the effective stress, $A$ and $B$ are material creep constants, and $n$ and $m$ are the material creep exponents. Creep is permissible with the following options only (I-1):

- isotropic strain hardening, IPLST = 2 — INACTIVE
- kinematic strain hardening, IPLST = 3 — INACTIVE

Assigning creep properties to a material does not automatically cause their effects to be included in an analysis. Creep effects are controlled by load system B load factor data (see “C-1 Load Multipliers” on page 11-11).

<table>
<thead>
<tr>
<th>ACO</th>
<th>BCO</th>
<th>M</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>creep constant $A$</td>
<td>creep constant $B$</td>
<td>creep exponent $m$</td>
<td>creep exponent $n$</td>
</tr>
</tbody>
</table>

- NTAM (B-3) number of entries in the Material Table
- NTAMT (B-3) number of entries in the Mount Element Table
- NTAB (B-3) number of entries in the Cross Section Table
- NTAW (B-3) number of entries in the Wall Fabrication Table
- NTAP (B-3) User Parameters flag
- NUNITS (B-2) number of shell units

if (NTAM materials have been defined) then
elseif (NTAMT > 0) then go to I-4a
elseif (NTAB > 0) then go to J-1
elseif (NTAW > 0) then go to K-1
elseif (NTAP > 0) then go to L-1
elseif (NUNITS > 0) then go to M-1
else follow instructions at end of R-3
else return to I-1
I-4a Mount Element Table Size

The Mount Element provides the capability of defining a generalized one dimensional force acting between two points in space with properties defined by the user. The force generated by the mount is directed along the line defined by the two points, like a “nonlinear spring.” These two points in turn are connected to nodes by rigid links, as shown in Figure 14.1 “E110 Mount element” on page 14-6, so that the net result will be, in general, both moments and forces at the structural nodes. If there are no rigid links, the mount element is presumed to be “pinned” to the two nodes. The stiffness and damping characteristics of the mount element are nonlinear functions of the axial relative displacement and axial relative velocity and are described by either STAGS input records or by a user-written subroutine.

Because of the action of the rigid links, the element force vectors and the stiffness and damping matrices computed for the E110 Mount Element will contain contributions for all six degrees of freedom at the nodes N1 and N2.

Mount reactions, or forces, that vary nonlinearly with both relative displacement and velocity between the ends of the mount are described best in tabular form that reflects how they were acquired—load tests at constant velocities with forces measured at prescribed displacements. This can be visualized as in Table 5.1 “Force-displacement-velocity profile” on page 5-66.

The I-4 records described in this section allow the user to define a force-displacement-velocity profile in tabular form for each mount with different behavior. The mounts themselves refer to the tables by the table identifier IMNT (below); these must be ordered consecutively.

The I-4 records described in this section also allow the user to define a penalty-function table that can be used with E810, E820 and/or E830 contact elements. In this case, the table must specify a stiffness parameter as a function of displacement. STAGS uses this penalty function to calculate forces and stiffnesses acting to enforce (non-penetration) compatibility conditions when contact occurs. The displacement parameter in a penalty-function table is sometimes called the “gap” in descriptions elsewhere in this Manual. A positive value of displacement measures the penetration of one contact point, line or surface into the line or surface that it contacts. In a penalty-function table, therefore, displacement must start at zero and increase monotonically; and the so-called “force” function should be given monotonically increasing stiffness values.

The I-4a record is the first of the I-4a/b/c/d series that are input NTAMT times (B-3). This record contains the number of relative displacements NRD and the number of relative velocities NRV defined in records I-4b and I-4c respectively.
### IMNT NRD NRV

<table>
<thead>
<tr>
<th>IMNT</th>
<th>Mount element or penalty function table id; table entries must be defined consecutively</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRD</td>
<td>For a mount element table: number of relative displacements, $NRD \geq 1$; relative displacement is $</td>
</tr>
<tr>
<td>NRV</td>
<td>For a mount element table: number of relative velocities, $NRV \geq 1$; if $NRV = 1$, forces are independent of velocity; for a penalty function table, $NRV$ must be unity</td>
</tr>
</tbody>
</table>

*go to I-4b*
Table 5.1  Force-displacement-velocity profile

<table>
<thead>
<tr>
<th>$d_1$</th>
<th>$d_2$</th>
<th>$d_3$</th>
<th>$\ldots$</th>
<th>$d_{nrd}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$</td>
<td>$F_{11}$</td>
<td>$F_{12}$</td>
<td>$F_{13}$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$v_2$</td>
<td>$F_{21}$</td>
<td>$F_{22}$</td>
<td>$F_{23}$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$v_3$</td>
<td>$F_{31}$</td>
<td>$F_{32}$</td>
<td>$F_{33}$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$v_{nrv}$</td>
<td>$F_{nrv1}$</td>
<td>$F_{nrv2}$</td>
<td>$F_{nrv3}$</td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>

$v_i$ is relative velocity $i$, $d_j$ is relative displacement $j$

$F_{ij}$ is the force corresponding to $d_j$ at $v_i$

✔ The $d_j$, $j = 1, nrd$ relative displacements are defined on I-4b

✔ The $v_i$, $i = 1, nrv$ relative velocities are defined on I-4c

✔ I-4d is repeated $nrv$ times, once for each relative velocity

✔ I-4d record $i$ contains $(F_{ij}, j = 1, nrd)$, the complete set of relative forces associated with $v_i$, and comprises row $i$ of Table 5.1
I-4b Relative Displacement Vector

This record contains the NRD relative displacements (or penetrations) for which mount forces (or contact stiffnesses) will be provided in record I-4d. These displacements must be entered starting with the smallest value and increasing to the largest value. For a mount element table, a positive relative displacement implies extension of the nonlinear spring, and a negative displacement implies compression. The relative displacement is explicitly $|l| - |l_0|$, where $l_0$ and $l$ are the initial and final distances between the ends of the mount, respectively. For a penalty function table, a positive displacement corresponds to penetration when contact occurs, and a negative value is meaningless.

\[
\text{DISP}(j), \ j = 1, \ NRD
\]

\[
\text{DISP}(j) \quad \text{relative displacement (or penetration) } j
\]

go to I-4c


I-4C Relative Velocity Vector

This record contains the $NRV$ relative velocities for which mount forces will be provided in record. These velocities must entered starting with the smallest value and increasing to the largest value. A positive relative velocity implies extension of the mount, and a negative relative velocity implies compression.

$RVEL(i)$, $i = 1, NRV$

$RVEL(i)$  relative velocity $i$

go to I-4d
**I-4d Mount Force Matrix**

For a mount element table, this record contains the mount forces corresponding to each of the relative displacements for a particular relative velocity. The complete set of forces in the force-displacement-velocity profile can be visualized as in Table 5.1 “Force-displacement-velocity profile” on page 5-66.

For a penalty function table, this record contains the stiffness value corresponding to each of the penetrations given on record I-4b. These should be monotonically increasing. Best results are usually obtained when the *stiffness vs. penetration* curve is “S” shaped, with a “toe” at the origin.

<table>
<thead>
<tr>
<th>FORCE(j), j = 1, NRD</th>
</tr>
</thead>
</table>

**FORCE(j)**
- For a mount table: the *force* corresponding to relative displacement *j*
- For penalty function table: the *stiffness* corresponding to penetration *j*

Note that *I-4d* contains *NRD* entries and is repeated *NRV* times.

**NRV** (I-4a) number of relative velocities
**NTAMT** (B-3) number of entries in the Mount Element Table
**NGCP** (B-3) GCP-utilization flag
**NTAB** (B-3) number of entries in the Cross Section Table
**NTAW** (B-3) number of entries in the Wall Fabrication Table
**NTAP** (B-3) User Parameters flag
**NUNITS** (B-2) number of shell units

if (NRV I-4d records have been defined) then
  if (NTAMT mount tables have been defined) then
    if (NGCP > 0) then go to I-5a
    elseif (NTAB > 0) then go to J-1
    elseif (NTAW > 0) then go to K-1
    elseif (NTAP > 0) then go to L-1
    elseif (NUNITS > 0) then go to M-1
    else follow instructions at end of R-3
  else return to I-4a
else continue defining I-4d
**I-5a GCP Command Record**

This record is read if and only if the **NGCP** (GCP-utilization) flag is 1 on the **B-3** record. I-5a departs from **STAGS**’ traditional integer/real-data input-format conventions and uses the command-oriented conventions used in NASA’s COMET program, from which the original versions of the Generic Constitutive Processor (GCP) routines used in **STAGS** were taken. The first item on this record is a command “verb” that instructs **STAGS** to read and catalog data for a specific type of material or fabrication, or to stop reading such data and proceed to the next type of model-definition input that is required.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>INFO(j), j = 1, 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMAND connection</td>
<td>command verb instructing <strong>STAGS</strong> to read a specific type of material or wall-fabrication data, or to cease doing so; <strong>STAGS</strong> currently recognizes and responds to the following commands:</td>
</tr>
<tr>
<td>ISOELASTIC_MATERIAL</td>
<td>linear, elastic, isotropic material</td>
</tr>
<tr>
<td>ORTHOELAST_MATERIAL</td>
<td>linear, elastic, orthotropic material</td>
</tr>
<tr>
<td>PLANE_STRAIN_MATERIAL</td>
<td>elastic/plastic plane strain material</td>
</tr>
<tr>
<td>PLASTIC_WB_MATERIAL</td>
<td>mechanical-sublayer (White-Besseling) material</td>
</tr>
<tr>
<td>ORT_EL_BR_MATERIAL</td>
<td>linear orthotropic elastic brittle material</td>
</tr>
<tr>
<td>ABAQUS_UMAT_MATERIAL</td>
<td>user-defined material model developed for <strong>ABAQUS</strong></td>
</tr>
<tr>
<td>PDLAM_MATERIAL</td>
<td>Chang’s progressive damage model, using crack density state variable</td>
</tr>
<tr>
<td>SHM_MEMBRANE_MATERIAL</td>
<td>Stein-Hedgepeth-Miller membrane wrinkling model</td>
</tr>
<tr>
<td>NL_ORT_ELAST_MATERIAL</td>
<td>Orthotropic elastic model, with Hahn nonlinearity</td>
</tr>
<tr>
<td>SHELL_FABRICATION</td>
<td>shell fabrication</td>
</tr>
<tr>
<td>SOLID_FABRICATION</td>
<td>solid fabrication</td>
</tr>
<tr>
<td>END</td>
<td>end of GCP data</td>
</tr>
</tbody>
</table>

**INFO**
integer vector of supplementary information: the meanings and permitted values of entries in **INFO** depend on the command verb; see the descriptions for records I-6a, I-7a, I-8a, I-9a, I-10a, I-11a, I-12a, I-13a, I-14a, I-21a and I-22a for those details.
The command verb on record 1-5a is case-independent. Each of the material- and fabrication-specification commands can be abbreviated to anything from its shortest form to its complete form—as shown in the following table:

**Table 5.2 GCP Command Record Summary**

<table>
<thead>
<tr>
<th>GCP Material Type</th>
<th>GCP Command Complete Form</th>
<th>Shortest Form</th>
<th>Status</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISOELASTIC_MATERIAL</td>
<td>ISOEL</td>
<td>✔</td>
<td>Isotropic, linear elastic model</td>
<td></td>
</tr>
<tr>
<td>ORTHOELAST_MATERIAL</td>
<td>ORTHO</td>
<td>✔</td>
<td>Orthotropic, linear elastic model</td>
<td></td>
</tr>
<tr>
<td>PLASTIC_WB_MATERIAL</td>
<td>PLAST</td>
<td>✔</td>
<td>Isotropic, elasto-plastic White-Besseling model</td>
<td></td>
</tr>
<tr>
<td>ORT_EL_BR_MATERIAL</td>
<td>ORT_E</td>
<td>✔</td>
<td>Orthotropic, linear elastic model with brittle failure</td>
<td></td>
</tr>
<tr>
<td>SHAPE_MEM_MATERIAL</td>
<td>SHAPE</td>
<td>Planned</td>
<td>Shape memory alloy model</td>
<td></td>
</tr>
<tr>
<td>PLANE_STRAIN_MATERIAL</td>
<td>PLANE</td>
<td>Planned</td>
<td>Plane-strain-plasticity model</td>
<td></td>
</tr>
<tr>
<td>PDLAM_MATERIAL</td>
<td>PDLAM</td>
<td>✔</td>
<td>Chang’s progressive damage model, with crack density state variable</td>
<td></td>
</tr>
<tr>
<td>ABAQUS_UMAT_MATERIAL</td>
<td>ABAQUS_UMAT</td>
<td>✔</td>
<td>ABAQUS UMAT compatible model</td>
<td></td>
</tr>
<tr>
<td>SHM_MEMBRANE_MATERIAL</td>
<td>SHM_MEMB</td>
<td>Under development</td>
<td>Stein-Hedgepeth-Miller membrane wrinkling model</td>
<td></td>
</tr>
<tr>
<td>NL_ORT_ELAST_MATERIAL</td>
<td>NL_ORT_EL</td>
<td>Under development</td>
<td>Orthotropic elastic model, with Hahn nonlinear in-plane shear stress-strain relation</td>
<td></td>
</tr>
</tbody>
</table>

SHELL_FABRICATION SHELL ✔ Shell element fabrication
SOLID_FABRICATION SOLID ✔ Solid element fabrication

END END ✔ End of GCP input data

The 3-character ‘END’ command cannot be abbreviated.
The following example shows how I-5a and other GCP–related records might be used in a typical STAGS input deck:

```
PCATS_6_gcp: 6-layer pcats problem
0 0 1 0 0 0 -1  $ B-1
1  $ B-2
0 0 0 0 0 1  $ B-3
2 15  $ P-1
$ PLASTIC_WB_MATERIAL 1 1 1 2 0  $ I-5a matid,ngroups,nstates,onetwo
72000.0 0.3 0.0 0.0 6 0.0  $ I-9a E1, Gnu, Rho, Beta, Nx, T
0.00600 432.0 .00702 490.0,  $ I-9b {strain,stress} values
0.00787 510.0 .01008 530.0,  $ I-9b {strain,stress} values
0.01247 540.0 .01900 555.0  $ I-9b {strain,stress} values
$ SHELL_FABRICATION 1 1 1 0 0  $ I-5a fabid,nlayer,ipts,ishr,isym
1  $ I-21a matid
5  $ I-21b intshl
2.0  $ I-21c thkshl
0.0  $ I-21d angshl
$ END  $ I-5a cease
$ 5  $ M-1
0. 5. 0. 90. 100.  $ M-2
-1 0 0. 0. 1 1  $ M-5
410  $ N-1
3 0 3 4  $ P-1
100 100  $ P-2
1 0 0 0  $ Q-1
1 2  $ Q-2
-100. 3 3 0 1  $ Q-3
0. -1 1 1  $ Q-3
1 0 0 0 0 0  $ R-1
```

The first two I-5a records in this example can be abbreviated as

```
PLASTIC_WB   1 1 1 2 0  $ I-5a
SHELL_FAB    1 1 1  $ I-5a
```

or as

```
PLAST 1 1 1 2
SHELL 1 1 1
```

if the user wishes.
### Data Tables Model Input

**NTAB** (B-3) number of entries in the Cross Section Table

**NTAW** (B-3) number of entries in the Wall Fabrication Table

**NTAP** (B-3) User Parameters flag

**NUNITS** (B-2) number of shell units

```plaintext
if (COMMAND = 'ISOELASTIC_MATERIAL') then go to I-6a
elseif (COMMAND = 'ORTHOELAST_MATERIAL') then go to I-7a
elseif (COMMAND = 'PLANE_STRAIN_MATERIAL') then go to I-8a
elseif (COMMAND = 'PLASTIC_WB_MATERIAL') then go to I-9a
elseif (COMMAND = 'ORT_EL_BR_MATERIAL') then go to I-10a
elseif (COMMAND = 'PDLAM_MATERIAL') then go to I-11a
elseif (COMMAND = 'ABAQUS_UMAT_MATERIAL') then go to I-12a
elseif (COMMAND = 'SHM_MEMB_MATERIAL') then go to I-13a
elseif (COMMAND = 'NL_ORT_ELAST_MATERIAL') then go to I-14a
elseif (COMMAND = 'SHELL_FABRICATION') then go to I-21a
elseif (COMMAND = 'SOLID_FABRICATION') then go to I-22a
else (COMMAND = 'END') then
  if (NTAB > 0) then go to J-1
  elseif (NTAW > 0) then go to K-1
  elseif (NTAP > 0) then go to L-1
  elseif (NUNITS > 0) then go to M-1
  else follow instructions at end of R-3
else return to I-5a
endif
```
I-6a Linear Elastic Isotropic GCP Material

This record—which concludes the specification of properties for a linear, elastic, isotropic GCP material—is read when COMMAND on I-5a is ‘ISOELASTIC_MATERIAL’, or is an acceptable abbreviation of that character string (as discussed in the I-5a record description). Under these circumstances, the INFO vector on I-5a contains the following information:

INFO(1) = MATID — material identifier, in the GCP Materials Table  
INFO(2) = NGROUPS — number of material groups (must be 1, currently)  
INFO(3) = NSTATES — number of material states in each group (must be 1, currently)  
INFO(4) = not used, currently  
INFO(5) = not used, currently  
INFO(6) = not used, currently  
INFO(7) = not used, currently

The following record is required for specification of the properties of the linear, elastic, isotropic GCP material for each state within each group of states (a single record, currently):

E GNU RHO ALPHA BETA T M

E          elastic modulus  
GNU        Poisson’s ratio  
RHO        mass density  
ALPHA      coefficient of thermal expansion  
BETA       coefficient of hygroscopic expansion  
T          reference temperature  
M          reference moisture content


go to I-5a
I-7a Linear Elastic Orthotropic GCP Material

This record—which continues the specification of properties for a linear, elastic, orthotropic GCP material—is read when COMMAND on I-5a is ‘ORTHOELAST_MATERIAL’, or is an acceptable abbreviation of that character string (as discussed in the I-5a record description). Under these circumstances, the INFO vector on I-5a contains the following information:

<table>
<thead>
<tr>
<th>INFO(1)</th>
<th>MATID</th>
<th>material identifier, in the GCP Materials Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFO(2)</td>
<td>NGROUPS</td>
<td>number of material groups (must be 1, currently)</td>
</tr>
<tr>
<td>INFO(3)</td>
<td>NSTATES</td>
<td>number of material states in each group (1, currently)</td>
</tr>
<tr>
<td>INFO(4)</td>
<td></td>
<td>not used, currently</td>
</tr>
<tr>
<td>INFO(5)</td>
<td></td>
<td>not used, currently</td>
</tr>
<tr>
<td>INFO(6)</td>
<td></td>
<td>not used, currently</td>
</tr>
<tr>
<td>INFO(7)</td>
<td></td>
<td>not used, currently</td>
</tr>
</tbody>
</table>

The following record is required for specification of the properties of the linear, elastic, orthotropic GCP material for each state within each group of states (this is a single record, currently, and all entries must be specified):

|----|----|----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|---|---|

E1, E2, E3  elastic moduli (E₁, E₂ and E₃)
G12, G13, G23  shear moduli (G₁₂, G₁₃ and G₂₃)
P12, P13, P23  Poisson’s ratios (v₁₂, v₁₃ and v₂₃)
RHO  mass density
A1, A2, A3  coefficients of thermal expansion
B1, B2, B3  coefficients of hygroscopic expansion
T  reference temperature
M  reference moisture content

The complete C matrix, which relates strains to stresses, is given by
\[
[c] = \begin{bmatrix}
\delta(1 - \nu_{23}\nu_{32})E_1 \\
\delta(\nu_{12} + \nu_{13}\nu_{32})E_2 \\
\delta(\nu_{13} + \nu_{12}\nu_{23})E_3 \\
0 & 0 & 0 & G_{23} \\
0 & 0 & 0 & 0 & G_{13} \\
0 & 0 & 0 & 0 & 0 & G_{12}
\end{bmatrix}
\]

where

\[
v_{21} = \nu_{12} \cdot \frac{E_2}{E_1}
\]

\[
v_{31} = \nu_{13} \cdot \frac{E_3}{E_1}
\]

\[
v_{32} = \nu_{23} \cdot \frac{E_3}{E_2}
\]

and

\[
\delta = \frac{1}{(1 - \nu_{12}\nu_{21} - \nu_{23}\nu_{32} - \nu_{13}\nu_{31} - 2\nu_{21}\nu_{32}\nu_{13})}
\]

Note that the \(\nu_{ij}\) conventions here are not the same as on the I-2 record, described previously.
I-8a Plane-Strain-Plasticity GCP Material

INACTIVE

This record—which concludes the specification of properties for an elastic/plastic, plane-strain GCP material—is read when COMMAND on I-5a is ‘PLANE_STRAIN_MATERIAL’, or is an acceptable abbreviation of that character string (as discussed in the I-5a record description).

Under these circumstances, the INFO vector on I-5a contains the following information:

- INFO(1) = MATID — material identifier, in the GCP Materials Table
- INFO(2) = NGROUPS— number of material groups (must be 1, currently)
- INFO(3) = NSTATES — number of material states in each group (must be 1, currently)
- INFO(4) = not used, currently
- INFO(5) = not used, currently
- INFO(6) = not used, currently
- INFO(7) = not used, currently

The following record is required for specification of the properties of the elastic/plastic, plane-strain GCP material for each state within each group of states (a single record, currently):

E GNU RHO ALPHA BETA T M

- E elastic modulus
- GNU Poisson’s ratio
- RHO mass density
- ALPHA coefficient of thermal expansion
- BETA coefficient of hygroscopic expansion
- T reference temperature
- M reference moisture content

*go to I-5a*
I-9a Mechanical Sublayer Plasticity GCP Material

This record—which continues the specification of properties for a mechanical sublayer plasticity (White–Besseling) GCP material—is read when the COMMAND parameter on the I-5a record is ‘PLASTIC_WB_MATERIAL’, or is an acceptable abbreviation of that character string (as discussed in the I-5a record description). Under these circumstances, the INFO vector on I-5a contains the following information:

| INFO(1) = MATID | material identifier, in the GCP Materials Table |
| INFO(2) = NGROUPS | number of material groups (must be 1, currently) |
| INFO(3) = NSTATES | number of material states in each group (must be 1, currently) |
| INFO(4) = ONETWO | dimensionality flag: |
| | ONETWO = 1 — use one-dimensional theory |
| | ONETWO = 2 — use two-dimensional theory |
| INFO(5) = IFLAG | material behavior flag: |
| | IFLAG = 0 — elastic/plastic material behavior |
| | IFLAG = 1 — elastic/plastic, plane-strain material behavior |
| INFO(6) = not used, currently |
| INFO(7) = not used, currently |

The following record is required for specification of the properties of the mechanical sublayer (White–Besseling) plasticity material for each state within each group of states (a single record, currently) — each such record followed by an I-9b record containing the specified number of \((e,s) = (\text{strain}, \text{stress})\) points on the curve for that state. This is strikingly similar to the White-Besseling input that is described for the I-3 record.

\[
\begin{array}{cccccc}
E & \text{elastic modulus} & \text{GNU} & \text{Poisson’s ratio} & \text{RHO} & \text{mass density} \\
\text{ALPHA} & \text{coefficient of thermal expansion} & \text{NSUBS} & \text{number of points on the } \varepsilon - \sigma \text{ curve, excluding the origin,} & \text{which is defined to be } (\varepsilon, \sigma) = (0, 0) \\
T & \text{reference temperature} \\
\end{array}
\]

\text{go to I-9b}
**I-9b Stress-Strain Curve for a Given State**

The White-Besseling theory (or mechanical sublayer model) has its limitations, but it performs adequately for a moderate-strain analysis of metals. This theory is based on a piecewise linear stress-strain curve, described on the data records by definition of strain and stress at the corners. The slope of the stress-strain curve must decrease at each corner. The first point on this curve must agree with elastic behavior, or the elastic modulus times the first strain value.

![Stress-strain curve](image)

**Figure 5.2** Stress-strain curve.

(E(i), S(i), i = 1, NSUBS)

| E(i) | εi, the strain value corresponding to the ith point on the (ε, σ) curve; the origin, (ε, σ) = (0, 0), is not included as a data point; (E(1), S(1)) must be defined to satisfy the relation σ1 = E ε1; where ε1 = E(1), σ1 = S(1), and E = E (the modulus of elasticity on I-9a). |
| S(i) | σi, the stress value corresponding to the ith point on the (ε, σ) curve |

*go to I-5a*
**I-10a Linear Orthotropic Elastic Brittle GCP Material**

This record*—which continues the specification of properties for a linear orthotropic elastic brittle GCP material—is read when **COMMAND** on I-5a is ‘ORT_EL_BR_MATERIAL’, or is an acceptable abbreviation of that character string (as discussed in the description of the I-5a record). Under these circumstances, the **INFO** vector on I-5a contains the following information:

<table>
<thead>
<tr>
<th>INFO(1) = MATID</th>
<th>material identifier, in the GCP Materials Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFO(2) = NGROUPS</td>
<td>number of material groups (must be 1, currently)</td>
</tr>
<tr>
<td>INFO(3) = NSTATES</td>
<td>number of material states in each group (1, currently)</td>
</tr>
<tr>
<td>INFO(4)</td>
<td>not used, currently</td>
</tr>
<tr>
<td>INFO(5)</td>
<td>not used, currently</td>
</tr>
<tr>
<td>INFO(6)</td>
<td>not used, currently</td>
</tr>
<tr>
<td>INFO(7)</td>
<td>not used, currently</td>
</tr>
</tbody>
</table>

The following record is required for specification of the properties of the linear, orthotropic elastic brittle GCP material for each state within each group of states. This is a single record currently, and all 44 entries must be specified as floating–point (real) entries. These data items are stored in this order in an internal real array named **mpd** as the original input material data. Caution must be exercised when converting from the traditional **STAGS** material model (I-2 records) to the GCP approach with regard to the definition of Poisson’s ratio (compare the description of the I-2 record with that of the I-7a record). All strain and strength allowable values are stored as positive numbers.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>EPS1C</td>
<td>EPS1T</td>
<td>ESP2C</td>
<td>EPS2T</td>
<td>EPS6F</td>
<td>EPS3C</td>
<td>EPS3T</td>
<td>EPS4F</td>
<td>EPS5F</td>
<td>XC</td>
<td>XT</td>
<td>YC</td>
<td>YT</td>
<td>SXY</td>
<td>ZC</td>
<td>ZT</td>
<td>SYZ</td>
<td>SXZ</td>
</tr>
</tbody>
</table>

* This description of the I-10a input requirements is based on that given in [2].

---

*E1, E2, E3*  
elastic moduli (E<sub>1</sub>, E<sub>2</sub> and E<sub>3</sub>)

*G12, G13, G23*  
shear moduli (G<sub>12</sub>, G<sub>13</sub> and G<sub>23</sub>)

*P12, P13, P23*  
major Poisson’s ratios (\(\nu_{12}\), \(\nu_{13}\) and \(\nu_{23}\))

*RHO*  
mass density

*A1, A2, A3*  
coefficients of thermal expansion

*B1, B2, B3*  
coefficients of hygroscopic expansion

*T*  
reference temperature for thermal calculations
M  reference moisture content for hygroscopic calculations
EPS1C, EPS1T  compressive and tensile strain allowable values in the 1–direction (ε_{1c} and ε_{1T})
EPS2C, EPS2T  compressive and tensile strain allowable values in the 2–direction (ε_{2c} and ε_{2T})
EPS6F  shear strain allowable value in the 1–2 plane (γ_{12})
EPS3C, EPS3T  compressive and tensile strain allowable values in the 3–direction (ε_{3c} and ε_{3T})
EPS4F  shear strain allowable value in the 2–3 plane (γ_{23})
EPS5F  shear strain allowable value in the 1–3 plane (γ_{13})
XC, XT  strength allowable values in the 1–direction (X_C and X_T)
YC, YT  strength allowable values in the 2–direction (Y_C and Y_T)
SXY  shear strength allowable value in the 1–2 plane (S_{xy} or S)
ZC, ZT  strength allowable values in the 3–direction (Z_C and Z_T)
SYZ  shear strength allowable value in the 2–3 plane (S_{yz})
SXZ  shear strength allowable value in the 1–3 plane (S_{xz})
ALPHA  nonlinear shear stress-strain coefficient, α
F12  coupling coefficient for the Tsai–Wu failure polynomial, F_{12}
BETA  scale factor for degrading material properties (e.g., 0.1 or 10^{-6}); if BETA is positive, then recursively degrade; if it is negative, then degrade only once (i.e., multiply elastic property by β only once)
IFAIL  failure model identifier:
  0 = pass through failure routines, but do not allow failure to occur
  1 = maximum strain criteria
  2 = maximum stress criteria
  3 = Tsai–Wu failure polynomial
  4 = Hashin criteria
  5 = Chang–Chang criteria
  99 = subroutine USRFPF, provided by the user
IDGRD  material degradation model identifier:
  1 = ply discounting of single modulus for maximum stress and maximum strain criteria
  2 = ply discounting of normal and shear moduli for maximum stress and maximum strain criteria
  3 = ply discounting for Tsai–Wu polynomial (same as 1)
  4 = ply discounting for Hashin criteria
  5 = ply discounting for Chang–Chang criteria
  99 = subroutine USRDGD, provided by the user
**VISF0**  
artificial viscous damping factor $\zeta_0$—multiplying all damping terms when convergence problems occur:  

- $> 1$ increase damping when nonconvergence occurs  
- $< 1$ reduce damping when nonconvergence occurs  
- $= 0$ eliminate all artificial viscous damping  
- $= 1$ maintain uniform damping  

**VISF1**  
artificial viscous damping factor $\zeta_1$ associated with global damping; becomes active after damage is detected and remains active, independent of damage state; scaled by the longitudinal modulus, $E_{11}$ (that is, $\zeta_1 E_{11}$)  

**VISFF**  
artificial viscous damping factor $\zeta_f$ associated with local damping; becomes active after damage is detected, is held constant for 5 solution steps, and is then ramped linearly to zero over the next 5 steps; scaled by the longitudinal modulus, $E_{11}$ (that is, $\zeta_f E_{11}$)  

The artificial viscous damping factor at a material point after damage has been detected has the form  

$$
\zeta = \zeta_0^\alpha (\zeta_1 E_{11} + \zeta_f E_{11})
$$  

where the exponent $\alpha$ is determined by using the initial user-specified step increment $\Delta P_0$ (on the C-1 record of the case.bin input file for s2) and on the current step increment $\Delta P$; that is,  

$$
\alpha = \log((\Delta P_0) / (\Delta P))
$$  

where $\Delta P \leq \Delta P_0 / 10$ before activation. When convergence difficulties arise that cause the step size to be cut smaller than the initial value, the exponent $\alpha$ increases. If **VISF0** is greater than one, the artificial damping will increase; if **VISF0** is less than one, it will decrease; and if **VISF0** is equal to zero, artificial damping will not be applied.  

The complete C matrix, which relates strains to stresses, is given in Section I-7a. The user may specify either the strain or strength allowables or both. However, as a minimum, a zero must be entered for each unspecified value. If the strain allowables are given and the first strength allowable is zero, then all strength allowables are computed by multiplying the appropriate strain allowable value by its elastic modulus (or shear modulus in the case of shear). If the strength allowables are specified and the first strain allowable is zero, then all strain allowables are computed by dividing the appropriate strength allowable value by its elastic modulus (or shear modulus in the case of shear).
A sample input record is shown next:

```
ORT_EL_BR_MATERIAL  1 1 1 $ I-5a GCP COMMAND RECORD
20.2E+6 1.41E+6 1.41E+6 , $ I-10a E1 E2 E3=E2
0.81E6 0.81E6 0.40E6 , $ I-10a G12 G13=G12 G23
0.29 0.29 0.29 , $ I-10a NU12 NU13=NU12 NU23=NU12
1.5E-4 , $ I-10a Mass density (not weight)
0.0 0.0 0.0 , $ I-10a CTE’S
0.0 0.0 0.0 , $ I-10a Moisture Coefficients
0.0 0.0 , $ I-10a Ref. Temp and Moisture contents
0.0109 0.0108 0.0227 0.0014 0.0074 , $ I-10a E1C E1T E2C E2T GAM12F
0.0227 0.0014 0.0074 0.0074 , $ I-10a E3C E3T GAM23F GAM13F
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0, $ I-10a Xc,Xt,Yc,Yt,Sxy Zc,Zt,Syz,Sxz
0.8E-14 0.0 1.0E-6, $ I-10a Alpha F12 Beta
1 1 , $ I-10a Ifail Idgrd
1.0 1.0E-9 1.0E-4 $ I-10a Visf0 Visf1 Visff
```

Failure prediction models based on stresses are implemented in subroutines named FPFi which are called by GCP subroutine CS4F. The output from each routine is an array named `flag` of length `numcol`. For C1 shell elements (e.g., E410 shell elements), `numcol` is equal to 3; for C0 shell elements (e.g., E480 shell elements), `numcol` is equal to 5; for solid elements (e.g., E881, E883 and E885 solid elements), `numcol` is equal to 6.

The entries in the `flag` array are integers that have values that are equal to zero for no failure, positive for tensile-related failures, or negative for compression-related failures. Shear-related failures are always positive. For shell elements, the first entry, `flag(1)`, relates to the fiber direction; the second entry, `flag(2)`, relates to the direction transverse to the fibers (matrix); and the third entry, `flag(3)`, relates to inplane shear. The fourth and fifth entries, `flag(4)` and `flag(5)`, relate to transverse shear failures. The absolute value of each entry in the flag array corresponds to the solution step number at which that failure mode occurred. For solid elements, the first three entries in `flag` relate to the normal stress components; and the next three entries relate to the shear stress components.
The following five failure models are implemented in the current version of STAGS:

- **FPF1**—Failure detection based on maximum strain criteria ($\textit{FAIL}=1$). These criteria are not interacting and simply compare the inplane mechanical strain to the user-specified strain allowable value.

For shell elements, the failure modes here are:

Fiber tensile failure: $\frac{\varepsilon_{11}}{\varepsilon_{1c}} \geq 1$ when $\varepsilon_{11} > 0$; $\text{flag}(1) > 0$

Fiber compressive failure: $\frac{\varepsilon_{11}}{\varepsilon_{1c}} < -1$ when $\varepsilon_{11} < 0$; $\text{flag}(1) < 0$

Matrix tensile failure: $\frac{\varepsilon_{22}}{\varepsilon_{2c}} \geq 1$ when $\varepsilon_{22} > 0$; $\text{flag}(2) > 0$

Matrix compressive failure: $\frac{\varepsilon_{22}}{\varepsilon_{2c}} \leq -1$ when $\varepsilon_{22} < 0$; $\text{flag}(2) < 0$

Inplane shear failure: $\frac{\varepsilon_{13}}{(\gamma_{12})_a} \geq 1$; $\text{flag}(3) > 0$

Transverse shear failure: $\frac{\varepsilon_{13}}{(\gamma_{13})_a} \geq 1$; $\text{flag}(4) > 0$

Transverse shear failure: $\frac{\varepsilon_{23}}{(\gamma_{23})_a} \geq 1$; $\text{flag}(5) > 0$

For solid elements, the failure modes are:

Fiber tensile failure: $\frac{\varepsilon_{11}}{\varepsilon_{1c}} \geq 1$ when $\varepsilon_{11} > 0$; $\text{flag}(1) > 0$

Fiber compressive failure: $\frac{\varepsilon_{11}}{\varepsilon_{1c}} \leq -1$ when $\varepsilon_{11} < 0$; $\text{flag}(1) < 0$

Matrix tensile failure: $\frac{\varepsilon_{22}}{\varepsilon_{2c}} \geq 1$ when $\varepsilon_{22} > 0$; $\text{flag}(2) > 0$

Matrix compressive failure: $\frac{\varepsilon_{22}}{\varepsilon_{2c}} \leq -1$ when $\varepsilon_{22} < 0$; $\text{flag}(2) < 0$

Interlaminar normal tensile failure: $\frac{\varepsilon_{33}}{\varepsilon_{3c}} \geq 1$ when $\varepsilon_{33} > 0$; $\text{flag}(3) > 0$

Interlaminar normal compressive failure: $\frac{\varepsilon_{33}}{\varepsilon_{3c}} \leq -1$ when $\varepsilon_{33} < 0$; $\text{flag}(3) < 0$

Transverse shear failure: $\frac{\varepsilon_{23}}{(\gamma_{23})_a} \geq 1$; $\text{flag}(4) > 0$

Transverse shear failure: $\frac{\varepsilon_{13}}{(\gamma_{13})_a} \geq 1$; $\text{flag}(5) > 0$

Inplane shear failure: $\frac{\varepsilon_{13}}{(\gamma_{12})_a} \geq 1$; $\text{flag}(6) > 0$
FPF2—Failure detection based on maximum stress criteria (\texttt{IFAIL}=2). These criteria are not interacting and simply compare the inplane stress components to the user-specified stress/strength allowable values.

**For shell elements, the failure modes here are:**

- Fiber tensile failure: \[
\frac{\sigma_{11}}{X_t} \geq 1 \quad \text{when} \quad \sigma_{11} > 0; \quad \text{flag(1)} > 0
\]
- Fiber compressive failure: \[
\frac{\sigma_{11}}{X_c} \leq -1 \quad \text{when} \quad \sigma_{11} < 0; \quad \text{flag(1)} < 0
\]
- Matrix tensile failure: \[
\frac{\sigma_{22}}{Y_t} \geq 1 \quad \text{when} \quad \sigma_{22} > 0; \quad \text{flag(2)} > 0
\]
- Matrix compressive failure: \[
\frac{\sigma_{22}}{Y_c} \leq -1 \quad \text{when} \quad \sigma_{22} < 0; \quad \text{flag(2)} < 0
\]
- Inplane shear failure: \[
\frac{|\tau_{12}|}{S_{xy}} \geq 1; \quad \text{flag(3)} > 0
\]
- Transverse shear failure: \[
\frac{|\tau_{23}|}{S_{xz}} \geq 1; \quad \text{flag(4)} > 0
\]
- Transverse shear failure: \[
\frac{|\tau_{33}|}{S_{yz}} \geq 1; \quad \text{flag(5)} > 0
\]

**For solid elements, the failure modes are:**

- Fiber tensile failure: \[
\frac{\sigma_{11}}{X_t} \geq 1 \quad \text{when} \quad \sigma_{11} > 0; \quad \text{flag(1)} > 0
\]
- Fiber compressive failure: \[
\frac{\sigma_{11}}{X_c} \leq -1 \quad \text{when} \quad \sigma_{11} < 0; \quad \text{flag(1)} < 0
\]
- Matrix tensile failure: \[
\frac{\sigma_{22}}{Y_t} \geq 1 \quad \text{when} \quad \sigma_{22} > 0; \quad \text{flag(2)} > 0
\]
- Matrix compressive failure: \[
\frac{\sigma_{22}}{Y_c} \leq -1 \quad \text{when} \quad \sigma_{22} < 0; \quad \text{flag(2)} < 0
\]
- Interlaminar normal tensile failure: \[
\frac{\sigma_{33}}{Z_t} \geq 1 \quad \text{when} \quad \sigma_{33} > 0; \quad \text{flag(3)} > 0
\]
- Interlaminar normal compressive failure: \[
\frac{\sigma_{33}}{Z_c} \leq -1 \quad \text{when} \quad \sigma_{33} < 0; \quad \text{flag(3)} < 0
\]
- Transverse shear failure: \[
\frac{|\tau_{23}|}{S_{yz}} \geq 1; \quad \text{flag(4)} > 0
\]
- Transverse shear failure: \[
\frac{|\tau_{33}|}{S_{xz}} \geq 1; \quad \text{flag(5)} > 0
\]
- Inplane shear failure: \[
\frac{|\tau_{12}|}{S_{xy}} \geq 1; \quad \text{flag(6)} > 0
\]
• FPF3—Failure detection based on the Tsai–Wu failure polynomial [3] (IFAIL=3). This criterion is interacting but does not identify a mode of failure (e.g., fiber failure, matrix failure). Most of the polynomial coefficients are computed based on user-specified strength allowable values—with the exception of \( F_{12} \), which must be supplied by the user as well. The complete polynomial is given by:

\[
F_1 \sigma_{11} + F_2 \sigma_{22} + F_3 \sigma_{33} + F_{11} \sigma_{11}^2 + F_{22} \sigma_{22}^2 + F_{33} \sigma_{33}^2 + 2F_{12} \sigma_{11} \sigma_{22} + 2F_{13} \sigma_{11} \sigma_{33} + 2F_{23} \sigma_{22} \sigma_{33} + \\
F_{44} \sigma_{23}^2 + F_{55} \sigma_{13}^2 + F_{66} \sigma_{12}^2 \geq 1.0
\]

where

\[
F_1 = \frac{1}{X_i - X_c}, \\
F_2 = \frac{1}{Y_i - Y_c}, \\
F_3 = \frac{1}{Z_i - Z_c}, \\
F_{11} = \frac{1}{X_i X_c}, \\
F_{22} = \frac{1}{Y_i Y_c}, \\
F_{33} = \frac{1}{Z_i Z_c}, \\
F_{44} = \frac{1}{S_{yz}}, \\
F_{55} = \frac{1}{S_{zx}}, \\
F_{66} = \frac{1}{S_{xy}}, \\
F_{12} = -\frac{1}{2} \frac{1}{\sqrt{X_i X_c Y_i Y_c}}, \\
F_{13} = -\frac{1}{2} \frac{1}{\sqrt{X_i X_c Z_i Z_c}}, \\
F_{23} = -\frac{1}{2} \frac{1}{\sqrt{Y_i Y_c Z_i Z_c}}.
\]

In this model, only a single value is checked to see whether failure occurred or not. In order to have set the failure flags used in the material degradation models, the approach used by Reddy and Reddy [4] is adopted. This approach determines the relative contributions of each stress component to the failure polynomial and if the contribution for that component is a dominant component, then the corresponding
failure flag is made nonzero. Half the cross term is considered in both the $\sigma_{11}$ term and in the $\sigma_{22}$ term. For shell elements, the coefficients $F_3$, $F_{33}$, $F_{13}$ and $F_{23}$ are zero because of the plane stress assumptions.

- FPF4—Failure detection based on the Hashin criteria [5] ($\text{FAIL}=4$). These criteria identify the mode of failure using specified stress/strength allowable values.

For shell elements, the failure modes here are:

- Fiber tensile failure: $\left( \frac{\sigma_{11}}{X_f} \right)^2 + \left( \frac{\sigma_{12}}{S_{xy}} \right)^2 \geq 1$ when $\sigma_{11} > 0$; $\text{flag}(1) > 0$
- Fiber compressive failure: $\frac{\sigma_{11}}{X_c} \leq -1$ when $\sigma_{11} < 0$; $\text{flag}(1) < 0$
- Matrix tensile failure: $\left( \frac{\sigma_{22}}{Y_f} \right)^2 + \left( \frac{\sigma_{12}}{S_{xy}} \right)^2 \geq 1$ when $\sigma_{22} > 0$; $\text{flag}(2) > 0$
- Matrix compression model: $\left( \frac{\sigma_{22}}{2S_{xy}} \right)^2 + \left( \frac{Y_c}{2S_{xy}} \right)^2 - 1 \sigma_{22} + \left( \frac{\sigma_{12}}{S_{xy}} \right)^2 \geq 1$ when $\sigma_{22} < 0$; $\text{flag}(2) < 0$
- Inplane shear failure: $\left( \frac{\tau_{12}}{S_{xy}} \right) > 0$; $\text{flag}(3) > 0$
- Transverse shear failure: $\left( \frac{\tau_{13}}{S_{xz}} \right) > 0$; $\text{flag}(4) > 0$
- Transverse shear failure: $\left( \frac{\tau_{23}}{S_{yz}} \right) > 0$; $\text{flag}(5) > 0$

For solid elements, the failure modes are:

- Fiber tensile failure: $\left( \frac{\sigma_{11}}{X_f} \right)^2 + \left( \frac{\sigma_{12}}{S_{xy}} \right)^2 \geq 1$ when $\sigma_{11} > 0$; $\text{flag}(1) > 0$
- Fiber compressive failure: $\frac{\sigma_{11}}{X_c} \leq -1$ when $\sigma_{11} < 0$; $\text{flag}(1) < 0$
- Matrix tensile failure: $\left( \frac{\sigma_{22}}{Y_f} \right)^2 + \left( \frac{\sigma_{12}}{S_{xy}} \right)^2 \geq 1$ when $\sigma_{22} > 0$; $\text{flag}(2) > 0$
- Matrix compression model: $\left( \frac{\sigma_{22}}{2S_{xy}} \right)^2 + \left( \frac{Y_c}{2S_{xy}} \right)^2 - 1 \sigma_{22} + \left( \frac{\sigma_{12}}{S_{xy}} \right)^2 \geq 1$ when $\sigma_{22} < 0$; $\text{flag}(2) < 0$
- Interlaminar normal tensile failure: $\sigma_{33} \geq 1$ when $\sigma_{33} > 0$; $\text{flag}(3) > 0$
- Interlaminar normal compressive failure: $\sigma_{33} \leq 1$ when $\sigma_{33} < 0$; $\text{flag}(3) < 0$
- Transverse shear failure: $\left( \frac{\tau_{23}}{S_{yz}} \right) \geq 1$; $\text{flag}(4) > 0$
- Transverse shear failure: $\left( \frac{\tau_{13}}{S_{xz}} \right) \geq 1$; $\text{flag}(5) > 0$
- Inplane shear failure: $\left( \frac{\tau_{12}}{S_{xy}} \right) \geq 1$; $\text{flag}(6) > 0$
• FPF5—Failure detection based on the Chang–Chang criteria [6] ($\text{IFAIL}=5$). These criteria identify the mode of failure using specified stress/strength allowable values.

For shell elements, the failure models here are:

Fiber breakage and fiber–matrix shearing model:

\[
\left( \frac{\sigma_{11}}{X_f} \right)^2 + \bar{\tau} \geq 1 \quad \text{when } \sigma_{11} > 0; \quad \text{flag}(1) > 0 \text{ and flag}(3) > 0
\]

Fiber compressive failure: \(\frac{\sigma_{11}}{X_c} \leq -1\) when \(\sigma_{11} < 0\); \(\text{flag}(1) < 0\)

Matrix cracking model: \(\left( \frac{\sigma_{22}}{Y_f} \right)^2 + \bar{\tau} \geq 1\) when \(\sigma_{22} > 0\); \(\text{flag}(2) > 0\)

Matrix compression model: \(\left( \frac{\sigma_{22}}{2S} \right)^2 + \left( \frac{Y_f}{2S} \right)^2 \left[ 1 - \frac{\sigma_{22}}{Y_c} \right] \geq 1\) when \(\sigma_{22} < 0\); \(\text{flag}(2) < 0\)

where \(\bar{\tau} = \left( \frac{\sigma_{12}}{S} \right)^2 \left[ 1 + \frac{3}{2} \alpha G_{12} \sigma_{12}^2 \right] \left[ 1 + \frac{3}{2} \alpha G_{12} S^2 \right]\)

Inplane shear failure: \(\frac{\tau_{13}}{S_{xy}} \geq 1\); \(\text{flag}(3) > 0\)

Transverse shear failure: \(\frac{\tau_{13}}{S_{xz}} \geq 1\); \(\text{flag}(4) > 0\)

Transverse shear failure: \(\frac{\tau_{23}}{S_{yz}} \geq 1\); \(\text{flag}(5) > 0\)

For solid elements, the failure models are:

Fiber breakage and fiber–matrix shearing model:

\[
\left( \frac{\sigma_{11}}{X_f} \right)^2 + \bar{\tau} \geq 1 \quad \text{when } \sigma_{11} > 0; \quad \text{flag}(1) > 0 \text{ and flag}(6) > 0
\]

Fiber compressive failure: \(\frac{\sigma_{11}}{X_c} \leq -1\) when \(\sigma_{11} < 0\); \(\text{flag}(1) < 0\)

Matrix cracking model: \(\left( \frac{\sigma_{22}}{Y_f} \right)^2 + \bar{\tau} \geq 1\) when \(\sigma_{22} > 0\); \(\text{flag}(2) > 0\)

Matrix compression model: \(\left( \frac{\sigma_{22}}{2S} \right)^2 + \left( \frac{Y_f}{2S} \right)^2 \left[ 1 - \frac{\sigma_{22}}{Y_c} \right] \geq 1\) when \(\sigma_{22} < 0\); \(\text{flag}(2) < 0\)

where \(\bar{\tau} = \left( \frac{\sigma_{12}}{S} \right)^2 \left[ 1 + \frac{3}{2} \alpha G_{12} \sigma_{12}^2 \right] \left[ 1 + \frac{3}{2} \alpha G_{12} S^2 \right]\)

Interlaminar normal tensile failure: \(\frac{\sigma_{33}}{Z_t} \geq 1\) when \(\sigma_{33} > 0\); \(\text{flag}(3) > 0\)

Interlaminar normal compressive failure: \(\frac{\sigma_{33}}{Z_c} \leq -1\) when \(\sigma_{33} < 0\); \(\text{flag}(3) < 0\)

Transverse shear failure: \(\frac{\tau_{23}}{S_{yz}} \geq 1\); \(\text{flag}(4) > 0\)

Transverse shear failure: \(\frac{\tau_{13}}{S_{xz}} \geq 1\); \(\text{flag}(5) > 0\)
Inplane shear failure: \[ \frac{\tau_{12}}{S_{xy}} \geq 1 \] ; \hspace{1cm} \text{flag(6)} > 0

Material degradation models based on ply discounting for each of the failure models are implemented in subroutines named \text{DGRADi} that are called by GCP subroutine \text{CS4F}. At present, the five material degradation models implemented include:

- **DGRAD1**—Material degradation model for the maximum strain and maximum stress failure models based on the inplane values only (\text{IDGRD}=1).

  For shell elements, the material degradation rules here are:

  If \text{flag}(1) is nonzero, set \( E_{11} = \beta E_{11}, \ P_{12} = \beta v_{12} \) and \( P_{21} = \beta v_{21} \) where \( \beta \) is a user-specified value (\text{e.g.,} \ 10^{-6}) read on the \text{ORT_EL_BR_MATERIAL} input data.

  If \text{flag}(2) is nonzero, set \( E_{22} = \beta E_{22}, \ P_{12} = \beta v_{12} \) and \( P_{21} = \beta v_{21} \)

  If \text{flag}(3) is nonzero, set \( G_{12} = \beta G_{12} \)

  If \text{flag}(4) is nonzero, set \( G_{13} = \beta G_{13} \)

  If \text{flag}(5) is nonzero, set \( G_{23} = \beta G_{23} \)

  For solid elements, the material degradation rules are:

  **Fiber failure** (tension or compression) (\text{flag}(1) \neq 0):

  set \( C_{11} = \beta C_{11}, \ C_{12} = \beta C_{12} \) and \( C_{13} = \beta C_{13} \)

  **Matrix failure** (tension or compression) (\text{flag}(2) \neq 0):

  set \( C_{22} = \beta C_{22}, \ C_{21} = \beta C_{21} \) and \( C_{23} = \beta C_{23} \)

  **Interlaminar normal failure** (tension or compression) (\text{flag}(3) \neq 0):

  set \( C_{33} = \beta C_{33}, \ C_{31} = \beta C_{31} \) and \( C_{32} = \beta C_{32} \)

  **Interlaminar shear failure** (\text{flag}(4) \neq 0): set \( C'_{44} = \beta C_{44} \)

  **Interlaminar shear failure** (\text{flag}(5) \neq 0): set \( C'_{55} = \beta C_{55} \)

  **In-plane shear failure** (\text{flag}(6) \neq 0): set \( C'_{66} = \beta C_{66} \)

- **DGRAD2**—Also a material degradation model for the maximum strain and maximum stress failure models based on the inplane values only (\text{IDGRD}=2). This model also degrades the inplane shear modulus when either fiber or matrix failures are detected.

  For shell elements, the material degradation rules here are:

  If \text{flag}(1) is nonzero, set \( E_{11} = \beta E_{11}, \ G_{12} = \beta G_{12}, \ P_{12} = \beta v_{12} \) and \( P_{21} = \beta v_{21} \),

  where \( \beta \) is a user-specified value (\text{e.g.,} \ 10^{-6}) read on the \text{ORT_EL_BR_MATERIAL} input data.

  If \text{flag}(2) is nonzero, set \( E_{22} = \beta E_{22}, \ G_{12} = \beta G_{12}, \ P_{12} = \beta v_{12} \) and \( P_{21} = \beta v_{21} \)

  If \text{flag}(3) is nonzero, set \( G_{12} = \beta G_{12} \)
If \( \text{flag}(4) \) is nonzero, set \( \mathbf{G}_{13} = \beta \mathbf{G}_{13} \)

If \( \text{flag}(5) \) is nonzero, set \( \mathbf{G}_{23} = \beta \mathbf{G}_{23} \)

For solid elements, the material degradation rules are:

Fiber failure (tension or compression) (\( \text{flag}(1) \neq 0 \)):

\[
\begin{align*}
\mathbf{C}_{11} &= \beta \mathbf{C}_{11}, \\
\mathbf{C}_{12} &= \beta \mathbf{C}_{12}, \\
\mathbf{C}_{13} &= \beta \mathbf{C}_{13}
\end{align*}
\]

Matrix failure (tension or compression) (\( \text{flag}(2) \neq 0 \)):

\[
\begin{align*}
\mathbf{C}_{22} &= \beta \mathbf{C}_{22}, \\
\mathbf{C}_{21} &= \beta \mathbf{C}_{21}, \\
\mathbf{C}_{23} &= \beta \mathbf{C}_{23}
\end{align*}
\]

Interlaminar normal failure (tension or compression) (\( \text{flag}(3) \neq 0 \)):

\[
\begin{align*}
\mathbf{C}_{33} &= \beta \mathbf{C}_{33}, \\
\mathbf{C}_{31} &= \beta \mathbf{C}_{31}, \\
\mathbf{C}_{32} &= \beta \mathbf{C}_{32}
\end{align*}
\]

Interlaminar shear failure (\( \text{flag}(4) \neq 0 \)):

\[
\begin{align*}
\mathbf{C}_{44} &= \beta \mathbf{C}_{44}
\end{align*}
\]

Interlaminar shear failure (\( \text{flag}(5) \neq 0 \)):

\[
\begin{align*}
\mathbf{C}_{55} &= \beta \mathbf{C}_{55}
\end{align*}
\]

In-plane shear failure (\( \text{flag}(6) \neq 0 \)):

\[
\begin{align*}
\mathbf{C}_{66} &= \beta \mathbf{C}_{66}
\end{align*}
\]

• DGRAD3—Material degradation model for the Tsai–Wu failure model [3] based on the inplane values only (\( \text{IDGRD}=3 \)). The material degradation rules follow those given by Reddy and Reddy [4].

For shell elements, the material degradation rules here are:

If \( \text{flag}(1) \) is nonzero, set \( E_{11} = \beta E_{11}, \mathbf{P}_{12} = \beta \nu_{13} \) and \( \mathbf{P}_{21} = \beta \nu_{21} \) where \( \beta \) is a user–specified value (e.g., \( 10^{-6} \)) read on the ORT_EL_BR_MATERIAL input data.

If \( \text{flag}(2) \) is nonzero, set \( E_{22} = \beta E_{22}, \mathbf{P}_{12} = \beta \nu_{12} \) and \( \mathbf{P}_{21} = \beta \nu_{21} \)

If \( \text{flag}(3) \) is nonzero, set \( \mathbf{G}_{12} = \beta \mathbf{G}_{12} \)

If \( \text{flag}(4) \) is nonzero, set \( \mathbf{G}_{13} = \beta \mathbf{G}_{13} \)

If \( \text{flag}(5) \) is nonzero, set \( \mathbf{G}_{23} = \beta \mathbf{G}_{23} \)

For solid elements, the material degradation rules are:

Fiber failure (tension or compression) (\( \text{flag}(1) \neq 0 \)):

\[
\begin{align*}
\mathbf{C}_{11} &= \beta \mathbf{C}_{11}, \\
\mathbf{C}_{12} &= \beta \mathbf{C}_{12}, \\
\mathbf{C}_{13} &= \beta \mathbf{C}_{13}
\end{align*}
\]

Matrix failure (tension or compression) (\( \text{flag}(2) \neq 0 \)):

\[
\begin{align*}
\mathbf{C}_{22} &= \beta \mathbf{C}_{22}, \\
\mathbf{C}_{21} &= \beta \mathbf{C}_{21}, \\
\mathbf{C}_{23} &= \beta \mathbf{C}_{23}
\end{align*}
\]

Interlaminar normal failure (tension or compression) (\( \text{flag}(3) \neq 0 \)):

\[
\begin{align*}
\mathbf{C}_{33} &= \beta \mathbf{C}_{33}, \\
\mathbf{C}_{31} &= \beta \mathbf{C}_{31}, \\
\mathbf{C}_{32} &= \beta \mathbf{C}_{32}
\end{align*}
\]

Interlaminar shear failure (\( \text{flag}(4) \neq 0 \)):

\[
\begin{align*}
\mathbf{C}_{44} &= \beta \mathbf{C}_{44}
\end{align*}
\]

Interlaminar shear failure (\( \text{flag}(5) \neq 0 \)):

\[
\begin{align*}
\mathbf{C}_{55} &= \beta \mathbf{C}_{55}
\end{align*}
\]

In-plane shear failure (\( \text{flag}(6) \neq 0 \)):

\[
\begin{align*}
\mathbf{C}_{66} &= \beta \mathbf{C}_{66}
\end{align*}
\]
• DGRAD4—Material degradation model for the Hashin [5] failure model based on the inplane values only (IDGRD=4). This model also degrades the inplane shear modulus when either fiber or matrix failures are detected. There is no inplane shear failure mode (i.e., flag(3) will always be zero).

For shell elements, the material degradation rules here are:

If flag(1) is nonzero, set $E_{11} = \beta E_{11}$, $G_{12} = \beta G_{12}$, $P_{12} = \beta \nu_{12}$ and $P_{21} = \beta \nu_{21}$
where $\beta$ is a user–specified value (e.g., $10^{-6}$) read on the ORT_EL_BR_MATERIAL input data.

If flag(2) is nonzero, set $E_{22} = \beta E_{22}$, $G_{12} = \beta G_{12}$, $P_{12} = \beta \nu_{12}$ and $P_{21} = \beta \nu_{21}$

If flag(3) is nonzero, set $G_{12} = \beta G_{12}$

If flag(4) is nonzero, set $G_{13} = \beta G_{13}$

If flag(5) is nonzero, set $G_{23} = \beta G_{23}$

For solid elements, the material degradation rules are:

Fiber failure (tension or compression) (flag(1) ≠ 0):

set $C_{11} = \beta C_{11}$, $C_{12} = \beta C_{12}$ and $C_{13} = \beta C_{13}$

Matrix failure (tension or compression) (flag(2) ≠ 0):

set $C_{22} = \beta C_{22}$, $C_{21} = \beta C_{21}$ and $C_{23} = \beta C_{23}$

Interlaminar normal failure (tension or compression) (flag(3) ≠ 0):

set $C_{33} = \beta C_{33}$, $C_{31} = \beta C_{31}$ and $C_{32} = \beta C_{32}$

Interlaminar shear failure (flag(4) ≠ 0): set $C_{44} = \beta C_{44}$

Interlaminar shear failure (flag(5) ≠ 0): set $C_{55} = \beta C_{55}$

In-plane shear failure (flag(6) ≠ 0): set $C_{66} = \beta C_{66}$

• DGRAD5—Material degradation model for the Chang–Chang failure model [6] based on the inplane values only (IDGRD=5). This model also degrades the inplane shear modulus when either fiber or matrix failures are detected.

For shell elements, the material degradation rules here are:

If flag(1) is nonzero (fiber breakage), set $E_{11} = \beta E_{11}$, $E_{22} = \beta E_{22}$, $G_{12} = \beta G_{12}$, $P_{12} = \beta \nu_{12}$ and $P_{21} = \beta \nu_{21}$

where $\beta$ is a user–specified value (e.g., $10^{-6}$) read on the ORT_EL_BR_MATERIAL input data. Chang and Chang actually use an exponentially degrading value based on the area of the damage zone. These factors are not in the current version.

If flag(2) is nonzero and positive (matrix cracking), set $E_{22} = \beta E_{22}$,
$G_{12} = \beta G_{12}$, $P_{12} = \beta \nu_{12}$ and $P_{21} = \beta \nu_{21}$

If flag(2) is nonzero and negative (matrix compression), set $E_{22} = \beta E_{22}$,
$P_{12} = \beta \nu_{12}$ and $P_{21} = \beta \nu_{21}$

If flag(3) is nonzero (fiber-matrix shearing), set $E_{11} = \beta E_{11}$, $E_{22} = \beta E_{22}$,
$G_{12} = \beta G_{12}$, $P_{12} = \beta \nu_{12}$ and $P_{21} = \beta \nu_{21}$
If \( \text{flag}(4) \) is nonzero, set \( G_{13} = \beta G_{13} \)

If \( \text{flag}(5) \) is nonzero, set \( G_{23} = \beta G_{23} \)

For solid elements, the material degradation rules are:

Fiber failure (tension or compression) \((\text{flag}(1) \neq 0)\):

set \( C_{11} = \beta C_{11}, \ C_{12} = \beta C_{12} \) and \( C_{13} = \beta C_{13} \)

Matrix failure (tension or compression) \((\text{flag}(2) \neq 0)\):

set \( C_{22} = \beta C_{22}, \ C_{21} = \beta C_{21} \) and \( C_{23} = \beta C_{23} \)

Interlaminar normal failure (tension or compression) \((\text{flag}(3) \neq 0)\):

set \( C_{33} = \beta C_{33}, \ C_{31} = \beta C_{31} \) and \( C_{32} = \beta C_{32} \)

Interlaminar shear failure \((\text{flag}(4) \neq 0)\): set \( C_{44} = \beta C_{44} \)

Interlaminar shear failure \((\text{flag}(5) \neq 0)\): set \( C_{55} = \beta C_{55} \)

In-plane shear failure \((\text{flag}(6) \neq 0)\): set \( C_{66} = \beta C_{66} \)

The progressive failure analysis implementation in STAGS provides the historical material database for postprocessing. For this material type, there are 12 data items for \( C^1 \) shell elements and 14 data items for \( C^0 \) shell elements, for each material point—the current material properties \((E_{11}, E_{22}, E_{33}, G_{23}, G_{13}, G_{12}, \nu_{23}, \nu_{13} \text{ and } \nu_{12})\), and the failure flags \((3 \text{ or } 5)\). These values may be plotted on an element level by layer, with STAGS’ stapl postprocessing program.

\( \to \) go to 1-5a
I-11a PDLAM GCP Material

This record—which contains the specification of properties for a progressive damage laminated composite material model based on the crack density state variable model of Shahid and Chang and implemented as a GCP material (Material Type 7)—is read when COMMAND on I-5a is ‘PDLAM_MATERIAL’, or is an acceptable abbreviation of that character string (as discussed in the description of the I-5a record). Under these circumstances, the INFO vector on I-5a contains the following information:

INFO(1) = MATID — material identifier, in the GCP Materials Table
INFO(2) = NGROUPS — number of material groups (must be 1, currently)
INFO(3) = NSTATES — number of material states (must be 1, currently)
INFO(4) = not used, currently
INFO(5) = not used, currently
INFO(6) = not used, currently
INFO(7) = not used, currently

The following record is required for specification of the properties of the PDLAM material model for each state within each group of states (this is currently a single 36-item real-data record, and all entries must be specified). These data items are stored, in this order, in an internal array named mpd as the original input material data. These data can be on a single line or on multiple lines with a comma denoting continuation of the record.

<table>
<thead>
<tr>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>G12</th>
<th>G13</th>
<th>G23</th>
<th>P12</th>
<th>P13</th>
<th>P23</th>
</tr>
</thead>
<tbody>
<tr>
<td>RHO</td>
<td>A1</td>
<td>A2</td>
<td>A3</td>
<td>B1</td>
<td>B2</td>
<td>B3</td>
<td>T</td>
<td>M</td>
</tr>
<tr>
<td>XC</td>
<td>XT</td>
<td>YC</td>
<td>SXZ</td>
<td>SYZ</td>
<td>ALPHA</td>
<td>CURET</td>
<td>G1C</td>
<td>G2C</td>
</tr>
<tr>
<td>VISF0</td>
<td>VISF1</td>
<td>VISFF</td>
<td>DPHI0</td>
<td>DELTA</td>
<td>DMIN</td>
<td>ETA</td>
<td>ETA</td>
<td>BETAC</td>
</tr>
</tbody>
</table>

E1, E2, E3  elastic moduli for each material direction (E_{11}, E_{22} and E_{33})
G12, G13, G23 elastic shear moduli (G_{12}, G_{13} and G_{23})
P12, P13, P23 major Poisson’s ratios (\nu_{12}, \nu_{13} and \nu_{23})
RHO  mass density, \rho
A1, A2, A3 coefficients of thermal expansion
B1, B2, B3 coefficients of hygroscopic expansion
T  reference temperature for thermal calculations
M  reference moisture content for hygroscopic calculations
XC  compression strength allowable values in the 1–direction (X_c)
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XT</td>
<td>Tensile strength allowable value in the 1–direction (X&lt;sub&gt;t&lt;/sub&gt;)</td>
</tr>
<tr>
<td>YC</td>
<td>Compression strength allowable value in the 2–direction (Y&lt;sub&gt;c&lt;/sub&gt;)</td>
</tr>
<tr>
<td>SYZ</td>
<td>Transverse shear strength allowable value in the 23–direction (S&lt;sub&gt;yz&lt;/sub&gt;)</td>
</tr>
<tr>
<td>SXZ</td>
<td>Transverse shear strength allowable value in the 13–direction (S&lt;sub&gt;xz&lt;/sub&gt;)</td>
</tr>
<tr>
<td>ALPHA</td>
<td>Nonlinear shear stress–strain coefficient, ( \alpha_0(G_{12})^3 ) where ( \alpha_0 ) is the usual coefficient having values in the range of 10&lt;sup&gt;-14&lt;/sup&gt;</td>
</tr>
<tr>
<td>CURET</td>
<td>Laminate cure temperature</td>
</tr>
<tr>
<td>G1C</td>
<td>Mode I fracture toughness, ( G_{Ic} )</td>
</tr>
<tr>
<td>G2C</td>
<td>Mode II fracture toughness, ( G_{IIc} )</td>
</tr>
<tr>
<td>VISF0</td>
<td>Global artificial damping factor independent of damage</td>
</tr>
<tr>
<td>VISF1</td>
<td>Global artificial damping coefficient, combined with longitudinal modulus</td>
</tr>
<tr>
<td>VISFF</td>
<td>Artificial damping coefficient after first fiber failure, combined with longitudinal modulus</td>
</tr>
<tr>
<td>DPHI0</td>
<td>Crack density increment, ( \Delta\phi )</td>
</tr>
<tr>
<td>DELTA</td>
<td>Fiber interaction zone, ( \delta )</td>
</tr>
<tr>
<td>DFMN</td>
<td>Lower limit of degradation factor</td>
</tr>
<tr>
<td>BETA</td>
<td>Fiber failure degradation rate, ( \beta )</td>
</tr>
<tr>
<td>ETA</td>
<td>Fiber/matrix shear-out failure degradation rate, ( \eta )</td>
</tr>
<tr>
<td>BETAC</td>
<td>Compression failure degradation rate, ( \beta_c )</td>
</tr>
</tbody>
</table>

The complete \( C \) matrix, which relates strains to stresses, is given in Section I-7a. Tension strength allowable value in the 2–direction (Y<sub>T</sub>) and the inplane shear strength allowable value (S) are dependent on the crack density state variable \( \phi \). These values as well as degraded reduced stiffness coefficients are computed \textit{a priori} using a stand-alone unit–cell plane–strain analysis of the laminate.

Failure detection is based on the Shahid–Chang criteria [7]. These criteria identify the mode of failure and utilize user–specified stress/strength allowable values.
Material degradation follows the formulation presented by Shahid and Chang using crack density \( \phi \) as a state variable. Prior to performing a nonlinear progressive failure analysis, the STAGS user firsts performs a plane–strain, unit–cell analysis of the laminate using the internal versions of subroutines PDCOMP and RDSE. This analysis yields the reduced material stiffness coefficients as a function of crack density for each layer in the laminate. Once failure is detected using the failure criteria listed previously, the degraded material stiffness coefficients for the corresponding crack density are used in the subsequent analysis step.

The present implementation of the PDLAM material formulation in STAGS assumes that a maximum of ten PDLAM material types can be present in a given model and that each shell fabrication using this material type uses a single PDLAM definition for all layers in that laminate. The current implementation of PDLAM is limited to symmetric laminates.

\[
\begin{align*}
\text{Fiber breakage mode: } & \quad \left(\frac{\sigma_{11}}{X_f}\right)^2 \geq 1 & \text{when } \sigma_{11} > 0 \\
\text{Fiber compression/shear mode: } & \quad \left(\frac{\sigma_{11}}{X_c}\right)^2 + \left(\frac{\sigma_{12}}{S(\phi)}\right)^2 \geq 1 & \text{when } \sigma_{11} < 0 \\
\text{Fiber–matrix shear-out mode: } & \quad \left(\frac{\sigma_{11}}{X_f}\right)^2 + \left(\frac{\sigma_{12}}{S(\phi)}\right)^2 \geq 1 & \text{when } \sigma_{11} > 0\\
& \quad \left(\frac{\sigma_{11}}{X_c}\right)^2 + \left(\frac{\sigma_{12}}{S(\phi)}\right)^2 \geq 1 & \text{when } \sigma_{11} < 0 \\
\text{Matrix cracking mode: } & \quad \left(\frac{\sigma_{22}}{Y_f(\phi)}\right)^2 + \left(\frac{\sigma_{12}}{S(\phi)}\right)^2 \geq 1 & \text{when } \sigma_{22} > 0 \\
\text{Matrix compression mode: } & \quad \left(\frac{\sigma_{22}}{Y_c}\right)^2 + \left(\frac{\sigma_{12}}{S(\phi)}\right)^2 \geq 1 & \text{when } \sigma_{22} < 0
\end{align*}
\]
I-12a ABAQUS UMAT GCP Material

This record—which contains the specification of properties for a user-defined material model developed for ABAQUS and implemented as a GCP material (Material Type 8)—is read when COMMAND on I-5a is ‘ABAQUS_UMAT_MATERIAL’, or is an acceptable abbreviation of that character string (as discussed in the description of the I-5a record). Under these circumstances, the INFO vector on I-5a contains the following information:

\[
\begin{align*}
\text{INFO}(1) &= \text{MATID} \quad \text{— material identifier, in the GCP Materials Table} \\
\text{INFO}(2) &= \text{NGROUPS} \quad \text{— number of material groups (must be 1, currently)} \\
\text{INFO}(3) &= \text{NSTATES} \quad \text{— number of material states (must be 1, currently)} \\
\text{INFO}(4) &= \text{not used, currently} \\
\text{INFO}(5) &= \text{not used, currently} \\
\text{INFO}(6) &= \text{not used, currently} \\
\text{INFO}(7) &= \text{not used, currently}
\end{align*}
\]

The following record is required for specification of the properties of the HKS/ABAQUS UMAT material model for each state within each group of states (this is currently a single 40-item real-data record, and all entries must be specified). These data items are stored, in this order, in an internal real array named \text{mpd} as the original input material data. These data can be on a single line or on multiple lines with a comma denoting continuation of the record.

\[
\text{PROPS(1)} \quad \text{PROPS(2)} \quad \ldots \quad \text{PROPS(40)}
\]

\[
\text{PROPS(i)} \quad \text{material property data items ( i=1, 2, ..., 40 )}
\]

See the Note, immediately after the following navigation instruction:

\[\text{go to I-5a}\]

**Note:** The main purpose of a user-defined material model is to return a trial stress state and trial values for the constitutive coefficients for the given strain state. Input values to an HKS/ABAQUS UMAT subroutine[8] include the strain state from the previous solution step (\text{STRAN}) and the increment in strain from that point to the current iteration cycle (\text{DSTRAN}). Then UMAT essentially returns the current strain state (\text{STRAN on exit}), trial stress state (\text{STRESS}) and trial constitutive coefficients (\text{DDSDDE}), and updated state variables set by the user (\text{STATEV}). These trial values are computed based on the strain state from the previous solution step, the increment in strain from that point to the present iteration cycle of the current solution step, and the trial constitutive coefficients corresponding to the present iteration cycle of the current solution step.
The calling sequence for UMAT based on the HKS/ABAQUS documentation is:

```fortran
SUBROUTINE UMAT ( STRESS, STATEV, DDSDDE, SSE, SPD, SCD,
1                  RPL, DDSDDT, DRPLDE, DRPLDT, STRAN, DSTRAN,
2                  TIME, DTIME, TEMP, DTEMP, PREDEF, DPRD,
3                  CMNAME, NDI, NSHR, NTENS, NSTATV, PROPS,
4                  NPROPS, COORDS, DROT, PNEWDT, CELENT, DFGRDO,
5                  DFGRD1, NOEL, NPT, LAYER, KSPT, KSTEP,
6                  KINC )
```

The functional status of the UMAT calling arguments for STAGS is given in Table 5.3 (at the end of this note). Not all of the HKS/ABAQUS UMAT arguments are active at this point. However, allocation of space and definition of variable type needs to be provided. The arguments in the UMAT calling sequence are defined as follows:

- **STRESS** – stress vector at the beginning of the solution step on entry and updated to the trial stress vector at the present iteration of the current step; dimensioned NTENS
- **STATEV** – array of state variables saved from the previous solution step at each material point and updated on exit to the trial values at the present iteration for the current solution step; defined by the UMAT developer; can be graphically displayed as a contour plot by stap1; dimensioned NSTATV
- **DDSDDE** – matrix of secant constitutive coefficients; dimensioned NTENS × NTENS
- **SSE** – strain energy density at a material point
- **SPD** – specific plastic dissipation at a material point
- **SCD** – specific creep or viscous dissipation at a material point
- **RPL** – volumetric heat generation per unit time
- **DDSDDT** – variation of the stress increments with respect to temperature; dimensioned NTENS
- **DRPLDE** – variation of RPL with respect to strain increment; dimensioned NTENS
- **DRPLDT** – variation of RPL with respect to temperature increment; dimensioned NTENS
- **STRAN** – total strain from the previous solution step on entry; total strain for present iteration of the current step on exit; dimensioned NTENS
- **DSTRAN** – strain increment from previous solution step to present iteration of the current solution step; dimensioned NTENS
- **TIME** – array with two entries; **TIME(1)** is the value of time at beginning of the current increment; **TIME(2)** is the value of total time at the beginning of the current increment
- **DTIME** – time increment or step size in a static analysis
• **TEMP** – temperature at the beginning of the increment
• **DTEMP** – increment of temperature
• **PREDEF** – array of interpolated values of predefined field variables at this material point at the start of this increment
• **DPRED** – array of increments of predefined field variables
• **CMNAME** – name of material within this UMAT subroutine
• **NDI** – number of direct stress terms: \(\text{NDI}=2\) for shell elements
• **NSHR** – number of shear stress terms: \(\text{NSHR}=1\) for \(C^1\) shell elements; \(\text{NSHR}=3\) for \(C^0\) shell elements
• **NTENS** – size of the stress or strain component array; dimensioned \(\text{NDI}+\text{NSHR}\)
• **NSTATV** – number of state variable to be archived at each material point; maximum of twenty variables (see the csumat.h header file)
• **PROPS** – material properties and parameters for this material model; defined and ordered by the UMAT developer; dimensioned \(\text{NPROPS}\)
• **NPROPS** – number of properties and values defined for this material model; maximum of forty variables
• **COORDS** – array containing the coordinates of this material point; dimensioned 3
• **DROT** – rotation increment matrix; dimensioned \(3 \times 3\)
• **PNEWDT** – ratio of suggested new time increment to the time increment being used
• **CELENT** – characteristic element length
• **DFGRD0** – array containing the deformation gradient at the beginning of the increment; dimensioned \(3 \times 3\)
• **DFGRD1** – array containing the deformation gradient at the end of the increment; dimensioned \(3 \times 3\)
• **NOEL** – element number for this material point
• **NPT** – surface integration point number within the \(\text{NOEL}\) element
• **LAYER** – layer number within the total laminate for this material point
• **KSPT** – section point within the current layer in the laminate; continuous numbering of the layer integration points throughout the laminate from bottom layer to top layer
• **KSTEP** – ABAQUS analysis step number; typically constant and equal to unity
• **KINC** – ABAQUS solution increment number in the analysis; STAGS step number

**STAGS** uses the variable **KSTEP** as a controlling flag to compute the initial constitutive coefficients (\(\text{KSTEP}=1\)), to perform a complete pass through UMAT (\(\text{KSTEP}=1\)), or to compute the current stress state in a post-processing step (\(\text{KSTEP}=2\)). This use is somewhat different than how **HKS/ABAQUS** uses the variable **KSTEP**. **STAGS** uses the variable **KINC** as the solution step number similar to the “time” increment number of **HKS/ABAQUS** (i.e., pseudo-time step). The linear solution always corresponds to a value of zero for **KINC**, and special provisions are included so that only linear elastic constitutive coefficients and stresses are computed.
### Table 5.3 Implementation Status of ABAQUS UMAT Variables in STAGS

#### Arguments to be Defined Within UMAT

<table>
<thead>
<tr>
<th>UMAT Argument Name</th>
<th>Active Now?</th>
<th>Not Available</th>
<th>Available in Future</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRESS (NTENS)</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATEV (NSTATV)</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSSDDE (NTENS,NTENS)</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSE</td>
<td>✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>SPD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RPL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DDSDDT (NTENS)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRPLDE (NTENS)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRPLDT</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Arguments Passed to UMAT as Information

<table>
<thead>
<tr>
<th>UMAT Argument Name</th>
<th>Active Now?</th>
<th>Not Available</th>
<th>Available in Future</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRAN (NTENS)</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSTRAN (NTENS)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME (1)</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME (2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DTIME</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEMP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DTEMP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PREDEF (1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DPRED (1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMNAME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDI</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSHR</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NTENS (=NDI+NSHR)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSTATV</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROPS (NPROPS)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NPROPS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COORDS (3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DROT (3,3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CELENT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DFGRD0 (3,3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DFGRD1 (3,3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOEL</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NPT</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAYER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KSPT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KSTEP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KINC</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Arguments that can be Updated within UMAT

<table>
<thead>
<tr>
<th>UMAT Argument Name</th>
<th>Active Now?</th>
<th>Not Available</th>
<th>Available in Future</th>
</tr>
</thead>
<tbody>
<tr>
<td>PNEWDT</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>
I-13a SHM-Membrane GCP Material

This record—which contains the specification of properties for the Stein-Hedgepeth-Miller membrane wrinkling model, implemented as a GCP material (Material Type 9)—is read when COMMAND on I-5a is ‘SHM_MEMBRANE_MATERIAL’, or is an acceptable abbreviation of that character string (as discussed in the I-5a record description). Under these circumstances, the INFO vector on I-5a contains the following information:

\[
\begin{align*}
\text{INFO}(1) &= \text{MATID} \quad \text{— material identifier, in the GCP Materials Table} \\
\text{INFO}(2) &= \text{NGROUPS} \quad \text{— number of material groups (must be 1, currently)} \\
\text{INFO}(3) &= \text{NSTATES} \quad \text{— number of material states in each group (must be 1, currently)} \\
\text{INFO}(4) &= \text{not used, currently} \\
\text{INFO}(5) &= \text{not used, currently} \\
\text{INFO}(6) &= \text{not used, currently} \\
\text{INFO}(7) &= \text{not used, currently}
\end{align*}
\]

The following record is required for specification of the nine properties of the Stein-Hedgepeth-Miller membrane wrinkling material model for each state within each group of states (this is a single record, currently, and all entries must be specified). These data items are stored, in this order, in an internal array named mpd as the original input material data. These data can be on a single line or on multiple lines with a comma denoting continuation of the record.

\[
\begin{align*}
\text{E GNU RHO ALPHA BETA T M PENLTY IWRINK ISTATE}
\end{align*}
\]

\[
\begin{align*}
\text{E} & \quad \text{elastic modulus} \\
\text{GNU} & \quad \text{Poisson’s ratio, } \nu \\
\text{RHO} & \quad \text{mass density, } \rho \\
\text{ALPHA} & \quad \text{coefficient of thermal expansion} \\
\text{BETA} & \quad \text{coefficient of hygroscopic expansion} \\
\text{T} & \quad \text{reference temperature} \\
\text{M} & \quad \text{reference moisture content} \\
\text{PENLTY} & \quad \text{penalty factor for the slack state} \\
\text{IWRINK} & \quad \text{wrinkling criteria:} \\
1 & \quad \text{Wong principal stress criteria} \\
2 & \quad \text{Wong principal strain criteria} \\
3 & \quad \text{Wong mixed or combined principal stress-strain criteria} \\
4 & \quad \text{Adler principal stress criteria} \\
5 & \quad \text{Adler principal strain criteria} \\
6 & \quad \text{Adler mixed or combined principal stress-strain criteria}
\end{align*}
\]
ISTATE  
initial membrane state for each iteration:

0 = initially always taut
1 = defined from previous converged solution

The complete C matrix, which relates strains to stresses, is given in Section I-7a. This matrix is changed during the computations, depending on the membrane state (taut, slack or wrinkled).

A sample input record for this material model is shown next:

```
SHM_MEMB 1 1 1  $ I-5a   GCP command record
3530.0   ,  $ I-13a  Young’s modulus, N/mm^2
0.30     ,  $ “     Poisson’s ratio
1.50E-6  ,  $ “     mass density, kg/mm^3
4*0.0    ,  $ “     ALPHA, BETA,T & M
1.00E-6  ,  $ “     penalty factor for slack state
3        ,  $ “     Wong’s mixed criteria
1                         $ “     initial state from previous step
```

See the Note, immediately after the following navigation instruction:

→  go to I-5a

Note: The basic concept behind the Stein-Hedgepeth-Miller model [9,10] is to examine points within the thin membrane structure and determine the current stress state. Since the structure is ultra-thin, bending is often neglected and the stress state is assumed constant through the thickness at a given planar coordinate. Denoting the planar deformation field as the xy-plane and given the state of strain ($e_{xx}, e_{yy}, \gamma_{xy}$) at a material point in that plane, the principal strains ($e_1, e_2$) are determined from:

\[
e_1, e_2 = \frac{e_{xx} + e_{yy}}{2} \pm \sqrt{\left(\frac{e_{xx} - e_{yy}}{2}\right)^2 + \left(\frac{\gamma_{xy}}{2}\right)^2}
\]  (5.1)

where the subscript 1 denotes major principal value and subscript 2 denotes the minor principal value. The orientation of the principal axes relative to x-axis is obtained from:

\[
\tan 2\alpha = \frac{\gamma_{xy}}{e_{xx} - e_{yy}}
\]  (5.2)
The stress state at that material point can be determined using the strain state and the generalized Hooke's law for an isotropic linear elastic material:

\[
\begin{pmatrix}
\sigma_{xx} \\
\sigma_{yy} \\
\sigma_{xy}
\end{pmatrix} =
\begin{bmatrix}
Q_{11} & Q_{12} & Q_{16} \\
Q_{12} & Q_{22} & Q_{26} \\
Q_{16} & Q_{26} & Q_{66}
\end{bmatrix}
\begin{pmatrix}
\varepsilon_{xx} \\
\varepsilon_{yy} \\
\gamma_{xy}
\end{pmatrix},
\]

where

\[
Q_{11} = Q_{22} = \frac{E}{1 - \nu^2}
\]
\[
Q_{12} = \frac{\nu E}{1 - \nu^2}
\]
\[
Q_{66} = G = \frac{E}{2(1 + \nu)}
\]
\[
Q_{16} = Q_{26} = 0
\]

and \(E\) is the elastic or Young's modulus and \(\nu\) is Poisson's ratio. Given this plane stress state, the principal stresses can be calculated according to:

\[
\sigma_1, \sigma_2 = \frac{\sigma_{xx} + \sigma_{yy}}{2} \pm \sqrt{\left(\frac{\sigma_{xx} - \sigma_{yy}}{2}\right)^2 + (\sigma_{xy})^2}
\]

where the subscript 1 denotes major principal value and subscript 2 denotes the minor principal value. The orientation of the principal axes relative to \(x\)-axis is obtained from:

\[
\tan 2\beta = \frac{2\sigma_{xy}}{\sigma_{xx} - \sigma_{yy}}
\]

The Stein-Hedgepeth-Miller model then examines these principal stress values and modifies the local constitutive relations for the membrane. Wong [11] and Adler [12] consider three criteria to define the membrane state: stress criteria; strain criteria; and a combined or mixed stress-strain criteria. While similar in concept, the actual criteria are slightly different and six different ones are implemented. The first three follow Wong [11] and the next three follow Adler [12]. These criteria are:
• Wong [11] stress criteria based on principal stresses ($\text{IWRINK}=1$)
  
  **Taut state:**  \( \sigma_2 > 0 \)
  
  **Slack state:**  \( \sigma_1 \leq 0 \) and  \( \sigma_2 \leq 0 \)
  
  **Wrinkled state:**  \( \sigma_1 > 0 \) and  \( \sigma_2 \leq 0 \)
  
• Wong [11] strain criteria based on principal strains ($\text{IWRINK}=2$)
  
  **Taut state:**  \( \varepsilon_2 > 0 \)
  
  **Slack state:**  \( \varepsilon_1 \leq 0 \) and  \( \varepsilon_2 \leq 0 \)
  
  **Wrinkled state:**  \( \varepsilon_1 > 0 \) and  \( \varepsilon_2 \leq 0 \)
  
• Wong [11] mixed stress-strain criteria based on both principal stresses and principal strains ($\text{IWRINK}=3$)
  
  **Taut state:**  \( \sigma_2 > 0 \)
  
  **Slack state:**  \( \varepsilon_1 \leq 0 \) and  \( \sigma_1 \leq 0 \)
  
  **Wrinkled state:**  \( \varepsilon_1 > 0 \) and  \( \sigma_2 \leq 0 \)
  
• Adler [12] stress criteria based on principal stresses ($\text{IWRINK}=4$)
  
  **Taut state:**  \( \sigma_2 > 0 \)
  
  **Slack state:**  \( \sigma_1 \leq 0 \)
  
  **Wrinkled state:**  \( \sigma_1 > 0 \) and  \( \sigma_2 \leq 0 \)
  
• Adler [12] strain criteria based on principal strains ($\text{IWRINK}=5$)
  
  **Taut state:**  \( \varepsilon_1 > 0 \) and  \( \varepsilon_2 > -v\varepsilon_1 \)
  
  **Slack state:**  \( \varepsilon_1 \leq 0 \)
  
  **Wrinkled state:**  \( \varepsilon_1 > 0 \) and  \( \varepsilon_2 \leq -v\varepsilon_1 \)
  
• Adler [12] mixed stress-strain criteria based on both principal stresses and principals strains ($\text{IWRINK}=6$)
  
  **Taut state:**  \( \sigma_2 > 0 \)
  
  **Slack state:**  \( \varepsilon_1 \leq 0 \)
  
  **Wrinkled state:**  \( \varepsilon_1 > 0 \) and  \( \sigma_2 \leq 0 \)

Of these criteria, the mixed stress-strain criteria appear to be the more commonly accepted criteria. Once the membrane state is determined at a material point, the constitutive model for the material point is set accordingly:
For the slack behavior, the constitutive coefficients are set to a penalty factor $\kappa$ (input parameter PENLTY) that is typically a very small number approaching zero and are given by:

$$Q_{ij}^{\text{slack}} = \kappa \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

(5.7)

For the taut behavior, the constitutive coefficients are the original plane stress values for a linear elastic, isotropic material given by:

$$Q_{ij}^{\text{taut}} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix}$$

(5.8)

For the wrinkled behavior, the constitutive coefficients are changed and are given by:

$$Q_{ij}^{\text{wrinkle}} = \frac{E}{4} \begin{bmatrix} 2(1+P) & 0 & Q \\ 0 & 2(1-P) & Q \\ Q & Q & 1 \end{bmatrix}$$

(5.9)

where $P = \cos 2\alpha = (\varepsilon_{xx} - \varepsilon_{yy})/(\varepsilon_1 - \varepsilon_2)$ and $Q = \sin 2\alpha = \gamma_{xy}/(\varepsilon_1 - \varepsilon_2)$.

The wrinkling process is considered to be an elastic process where wrinkles may form and then disappear if the local stress state at a material point would permit it.

This material model can be used with membrane elements (constraint all bending degrees of freedom to zero using the Q-records) and also with plate/shell elements. Within STAGS, membrane behavior is modeled by constraining all out-of-plane deformation degrees of freedom (transverse displacement and two bending rotations) using the loading records (Q-records). The E410 shell element uses the “drilling degree of freedom” to increase the order of the in-plane displacement field approximations; the E330 and E480 shell elements do not have this freedom and so it must also be constrained. Using this material model with plate/shell elements means that the local material stiffness terms are modified in the same way based on the wrinkling criteria and in addition, the bending stiffness terms are computed and used in the stiffness matrix evaluation.
I-14a Nonlinear Orthotropic Elastic GCP Material

This record—which continues the specification of properties for a nonlinear orthotropic elastic GCP material model based on the Hahn nonlinear in-plane shear stress-strain definition [13,14], implemented as a GCP material (Material Type 10)—is read when COMMAND on I-5a is ‘NL_ORT_ELAST_MATERIAL’, or is an acceptable abbreviation of that character string (as discussed in the I-5a record description). Under these circumstances, the INFO vector on I-5a contains the following information:

```
INFO(1) = MATID — material identifier, in the GCP Materials Table
INFO(2) = NGROUPS — number of material groups (must be 1, currently)
INFO(3) = NSTATES — number of material states in each group (1, currently)
INFO(4) = not used, currently
INFO(5) = not used, currently
INFO(6) = not used, currently
INFO(7) = not used, currently
```

The following record is required for specification of the nineteen properties of the nonlinear orthotropic elastic material model for each state within each group of states (this is a single record, currently, and all entries must be specified). These data items are stored, in this order, in an internal array named mpd as the original input material data. These data can be on a single line or on multiple lines with a comma denoting continuation of the record.

```
```

- **E1, E2, E3** elastic moduli (E₁, E₂ and E₃)
- **G12, G13, G23** shear moduli (G₁₂, G₁₃ and G₂₃)
- **P12, P13, P23** Poisson’s ratios (v₁₂, v₁₃ and v₂₃)
- **RHO** mass density, ρ
- **A1, A2, A3** coefficients of thermal expansion
- **B1, B2, B3** coefficients of hygroscopic expansion
- **T** reference temperature
- **M** reference moisture content
- **S6666** nonlinear parameter for in-plane shear stress-strain relations

The complete linear elastic C matrix, which relates strains to stresses, is given in Section I-7a.
A sample input record for this material model is shown next:

```
NL ORT_EL 1 1 1                     $ I-5a  GCP command record
19.00E+6 1.89E+6 1.89E+6 ,         $ I-14a  elastic moduli [psi]
0.93E+6 0.93E+6 0.93E+6 ,          $ "  shear moduli [psi]
0.38 0.38 0.38 ,                   $ "  Poisson's ratios
2.59E-5 ,                          $ "  mass density [slugs]
3*0.0 3*0.0 2*0.0 ,                $ "  CTEs, CMEs, T & M
1.00E-6 ,                          $ "  nonlinear shear parameter
```

See the Note, immediately after the following navigation instruction:

→ go to I-5a

**Note:** The basic concept behind the Hahn-Tsai [14] and Hahn [13] mode is to formulate a complementary strain energy density function and determine the strain-stress relations. From Hahn and Tsai [14], a unique solution for the in-plane shear strain-stress relations is derived to have the form of a cubic equation given by:

\[
e_6 = 2e_{12} = S_{66} \sigma_{12} + S_{6666} \sigma_{12}^3 = \frac{1}{G_{12}} \sigma_{12} + \frac{\alpha}{G_{12}} \sigma_{12}^3
\]  

(5.11)

A unique solution to the cubic equation exists if:

\[
\frac{de_6}{d\sigma_{12}} = S_{66} + 3S_{6666} \sigma_{12}^2 \neq 0
\]  

(5.12)

For real material systems, the coefficient \(S_{66}\) is a positive real number; hence a unique solution requires that \(S_{6666}\) be a real positive number greater than zero for all values of the shear stress. Hahn and Tsai [14] then conclude that there is only one real root of the nonlinear shear strain-stress equation, and the solution is denoted by:

\[
\sigma_{12} = \left[ \frac{1}{S_{66}} + f(e_6) \right] e_6
\]  

(5.13)

where \(f(e_6)\) is the real root of the cubic equation given by:
A general cubic equation of the form

\[ y^3 + \frac{3}{S_{66}}y^2 + \left( \frac{3}{S_{66}} + \frac{S_{66}}{S_{666}e_6} \right)y + \frac{1}{S_{66}} = 0 \]  

(5.15)

can be reduced to

\[ x^3 + ax + b = 0 \]  

(5.17)

by substituting \( y = x - \frac{p}{3} \) and noting the following definitions for the constants \( a \) and \( b \):

\[ a = \frac{1}{3}(3q - p^2) \quad \text{and} \quad b = \frac{1}{27}(2p^3 - 9pq + 27r) \]  

(5.18)

There will be one real root and two conjugate imaginary roots if

\[ c = \frac{b^2}{4} + \frac{a^3}{27} > 0 \]  

(5.19)

The solution for \( x \) (and therefore \( y \)) is then obtained in terms of \( A \) and \( B \):

\[ x = A + B, \quad \frac{A+B}{2} + \frac{A-B}{2} \sqrt[3]{3} \quad \text{and} \quad \frac{A+B}{2} - \frac{A-B}{2} \sqrt[3]{3} \]  

(5.20)

where

\[ A = \sqrt[3]{\frac{b}{2} + \sqrt[3]{c}} \quad \text{and} \quad B = \sqrt[3]{\frac{b}{2} - \sqrt[3]{c}} \]  

(5.21)

The solution to this cubic equation is calculated in subroutine CROOT (see croot.F in the GCP subdirectory). The coefficients \( (p, q \) and \( r \) are calling arguments and the real root is returned from the subroutine.

Within the GCP of STAGS, the material model is entered with a given strain state and returns trial stress values and, for nonlinear materials, trial constitutive coefficients. Thus, given the
value of the in-plane shear strain $\gamma_6 = \gamma_{12} = 2\varepsilon_{12}$, the linear elastic shear modulus $G_{12}$ and the nonlinear parameter $S_{6666}$, the in-plane shear stress can be determined by solving the cubic equation using the approach just described. The $S_{6666}$ term has units of the reciprocal of stress to the third power. That is, 

$$S_{6666} = \frac{\alpha}{G_{12}^3}$$

(5.22) 

and has a value on the order of $1 \times 10^{-15} \text{ (psi)}^{-3}$ or $10 \text{ (GPa)}^{-3}$ or $1 \times 10^{-8} \text{ (Mpa)}^{-3}$ and hence the order of $\alpha$ is in the 1000's. This term is an experimentally determined quantity from off-axis tension tests [14].
I-21a GCP Shell Fabrication Record

This record—which continues the specification of a shell fabrication in the GCP—is read when COMMAND on I-5a is ‘SHELL_FABRICATION’, or is an acceptable abbreviation of that character string (as discussed in the I-5a record description). Under these circumstances, the INFO vector on I-5a contains the following information:

- **INFO(1) = FABID** — shell fabrication identifier, in the GCP Fabrications Table:
  
  > 0 — use material and fabrication properties specified here for all elements referencing this FABID
  
  < 0 — call user-written subroutine USRFAB to determine if an element referencing this fabrication uses material and fabrication properties specified here or properties that are specified in USRFAB (see Chapter 12)

- **INFO(2) = NLAYER** — total number of material layers in the fabrication;
  
  \( 1 \leq NLAYER \leq 100 \)

- **INFO(3) = IPTS** — through-layer-integration-points flag:
  
  0 — integrate at 2 points (bottom & top) for each layer
  
  1 — specify number of through-layer points for each layer

- **INFO(4) = ISHR** — shear-factor-specification flag:
  
  0 — do not specify shear-correction factors
  
  1 — specify shear-correction factors

- **INFO(5) = ISYM** — layer–symmetry flag:
  
  0 — general layup; specify data for all of the layers of the layup (i.e., set \( NX = 1 \))
  
  1 — fabrication is symmetric through the thickness; specify data for the top layers—from layer number \( NX = NLAYER/2 + 1 \) to layer number \( NLAYER \) (which must be even);
  
  STAGS will then symmetrize the layup

- **INFO(6) = not used, currently**

- **INFO(7) = not used, currently**

Note:

For \( C^0 \) shell elements, the stress and strain components have the following orders:

\[
\sigma_1, \quad \sigma_2, \quad \tau_{12}, \quad \tau_{13}, \quad \tau_{23}
\]
\[
\varepsilon_1, \quad \varepsilon_2, \quad \gamma_{12}, \quad \gamma_{13}, \quad \gamma_{23}
\]

For \( C^1 \) shell elements, the stress and strain components have the following orders:

\[
\sigma_1, \quad \sigma_2, \quad \tau_{12}
\]
\[
\varepsilon_1, \quad \varepsilon_2, \quad \gamma_{12}
\]
MATID(j)  \( j = \text{NX}, \text{NLAYER} \)

MATID(j) material identifier (in the GCP Materials Table) for the \( j^{\text{th}} \) layer, where

\[
\text{NX} = 1 \quad (\text{when ISYM} = 0) \quad \text{or} \quad \text{NX} = \text{NLAYER}/2 + 1 \quad (\text{when ISYM} = 1)
\]

INFO(3) = IPTS  (I-5a) through-layer-integration-points flag

\[
\begin{align*}
\text{if} \quad \text{(IPTS > 0)} & \quad \text{go to I-21b} \\
\text{else} & \quad \text{go to I-21c}
\end{align*}
\]
I-21b GCP Shell Integration Points Record

This record—which continues the specification of a shell fabrication in the GCP—should be included if and only if \( \text{IPTS} = 1 \) on I-21a: it specifies the number of through-layer integration points to be used for each layer for which data are being specified.

\[
\text{INTSHL}(j) \quad j = \text{NX, NLAYER}
\]

\( \text{INTSHL}(j) \) number of integration points to be used for the \( j^{\text{th}} \) layer of the layup, where \( \text{NX} = 1 \) (when \( \text{ISYM} = 0 \)) or \( \text{NX} = \text{NLAYER}/2 + 1 \) (when \( \text{ISYM} = 1 \))

Note:

It is appropriate to set

\[
\begin{align*}
\text{INTSHL}_j &= 1 \quad \text{when any bending within a layer is assumed negligible} \\
\text{INTSHL}_j &= 0 \text{ or } 2 \quad \text{to integrate at two locations within each layer (at } \pm \frac{1}{\sqrt{3}} \frac{h_k}{2} \text{, on either side of the midsurface of that layer); this is exact when stress varies linearly over the layer; this is STAGS’ default value} \\
\text{INTSHL}_j &> 2 \quad \text{to integrate using Simpson’s 1/3 rule (odd values only):} \\
\int_a^b f(\xi) d\xi &\approx \frac{1}{3} (b - a) \left[ f(a) + 4f\left(\frac{a + b}{2}\right) + f(b) \right]
\end{align*}
\]

\( \text{go to I-21c} \)
I-21c GCP Shell Layer Thickness Record

This record—which continues the specification of a shell fabrication in the GCP—must be included: it specifies the thickness of each layer for which data are being specified.

\[ \text{THKSHL}(j) \quad j = \text{NX, N\text{LAYER}} \]

- thickness of the \( j \)th layer, where \( \text{NX} = 1 \) (when ISYM=0) or \( \text{NX} = \text{NLAYER}/2 + 1 \) (when ISYM = 1)

*go to I-21d*
**I-21d GCP Shell Layer Orientation Record**

This record—which continues the specification of a shell fabrication in the GCP—must be included: it specifies the fabrication orientation angle for each layer for which data are being specified.

**ANGSHL(j)  j = NX, NLAYER**

**ANGSHL(j)** fabrication orientation angle for the j\(^{\text{th}}\) layer, where \(\text{NX} = 1\) (when ISYM=0) or \(\text{NX} = (\text{NLAYER} + 1)/2\) (when ISYM=1)

\[\text{INFO}(4) = \text{ISHR} \quad (\text{I-5a})\]

shear-factor-specification flag

if \((\text{ISHR} > 0)\)  
**go to I-21e**
else \(\text{go to I-5a}\)
**I-21e Shear Factor Specification Record**

This record—which concludes the specification of a shell fabrication in the GCP—must be included when $\text{ISHR} = 1$ on the I-21a record: it specifies the two shear-correction factors to be for all layers of the current fabrication.

### SCF1  SCF2

<table>
<thead>
<tr>
<th>SCF1</th>
<th>SCF2</th>
</tr>
</thead>
<tbody>
<tr>
<td>first shear-correction factor; if $\text{ISHR} = 0$, STAGS sets $\text{SCF1} = 1.0$</td>
<td>second shear-correction factor; if $\text{ISHR} = 0$, STAGS sets $\text{SCF2} = 1.0$</td>
</tr>
</tbody>
</table>

**Note 1:** The nominal value for each of these parameters is $5/6$, for isotropic materials.

**Note 2:** Under the traditional STAGS approach for shell wall fabrications using K records, STAGS internally assumes that $\text{SCF1} = \text{SCF2} = 0.83333333 = 5/6$ when C$^0$ elements are used.

*go to I-5a*
I-22a GCP Solid Fabrication Record

This record—which continues the specification of a solid fabrication in the GCP—is read when COMMAND on I-5a is ‘SOLID_FABRICATION’, or is an acceptable abbreviation of that character string (as discussed in the I-5a record description). Under these circumstances, the INFO vector on I-5a contains the following information:

INFO(1) = FABID  —  solid fabrication identifier, in the GCP Fabrications Table:
> 0  —  use material and fabrication properties specified here for all solid elements referencing this FABID
< 0  —  call user-written subroutine USRFAB to determine if an element referencing this fabrication uses material and fabrication properties specified here or properties that are specified in USRFAB (see Chapter 12)

INFO(2) = MATID  —  material identifier, in the GCP Materials Table, for the GCP material used in this solid fabrication

INFO(3) = IANG  —  fabrication-orientation flag:
0  —  set orientation angles to zero
1  —  input orientation angles [in degrees]

INFO(4) = NIPZ  —  number of through–thickness integration points (1 or 2) (default = 2); NIPZ is meaningful only for the core components of E800-series sandwich elements

INFO(5) = not used, currently
INFO(6) = not used, currently
INFO(7) = not used, currently

Note:

For solid elements, the stress and strain components have the following orders:

\[ \sigma_1, \sigma_2, \sigma_3, \tau_{23}, \tau_{13}, \tau_{12} \]
\[ \varepsilon_1, \varepsilon_2, \varepsilon_3, \gamma_{23}, \gamma_{13}, \gamma_{12} \]
Figure 5.4  GCP Solid Fabrication

THICK

INFO(3) = IANG  (l-5a)  fabrication-orientation flag

if (IANG > 0)  go to l-22b
else  go to l-5a

THICK  element thickness
I-22b GCP Solid Fabrication Orientation Record

This record—which concludes the specification of a solid fabrication in the GCP—must be included if $\text{INFO}(3) = \text{IANG} = 1$ on the I-5a record: it specifies the orientation angle to be used for the current solid fabrication.

**ANGLE**

- **ANGLE**: orientation angle, in degrees (in the x-y plane; rotations are measured about the z-axis in a right-hand-rule sense)

  \( \text{go to I-5a} \)
**J-1 Cross-Section**

The records J-1–J-3 are included only if there are beam elements or stiffeners to be defined, \( NTAB > 0 \) (B-3). The record sequence is repeated \( NTAB \) times. These records define cross-section properties, including, via reference to the Material Table (I records), material properties. These data can be referenced in subsequent definitions of beam elements and stiffeners.

In the case of elastic deformation, the strain energy in a beam (stiffener) can be expressed in terms of a few parameters such as cross-sectional area and moments of inertia. Whenever a beam (stiffener) is defined by use of such data, we refer to its cross-section as being of the general type.

If plastic deformation is included or if thermal expansion varies nonlinearly within the cross-section, the total strain is still linear but the stresses are not. In that case the elastic strain energy in the beam (stiffener) depends on the actual shape of the cross-section. For such applications cross-sections of subelement type are defined. The cross-section is decomposed into a number of subelements, each sufficiently small so that thermal expansion and plastic strain may (as a good approximation) be considered constant within the subelement. Generally, then the moments of inertia of the subelements about their own axes are negligible. If more than \( NSUB \) subelements must be included, two or more beams (stiffeners) may be co-located (with the same effect as the inclusion of one cross-section with a larger number of subelements). Plasticity and thermal effects can be included in stiffeners of the subelement type only ( \( KCROSS = 2 \) or 3).

The subelement type of cross-section is useful even in elastic analysis without thermal effects. A T-stiffener, for example, can be defined as a subelement type cross-section with two rectangular subelements.

A beam (stiffener) cross-section is defined in \((\tilde{y}, \tilde{z})\) cross-section coordinates (see Section 4.1). When a beam (stiffener) element is defined, the user can specify an angle between the \( z' \) (element unit) or \( Z' \) (shell unit) axis and the cross-section \( \tilde{z} \) axis. An eccentricity, the \((y', z')\) (element unit) or \((Y', Z')\) (shell unit) coordinates at the origin of the \((\tilde{y}, \tilde{z})\) system, can also be specified (see Figure 6.5 on page 6-52 and Figure 8.1 on page 8-12).

An elastic Timoshenko beam capability can be used with the 200-series beams in STAGS. These beams include those automatically introduced when rings and stringers are called for on shell units discretized using the 300- and 400-series shell elements (see the description of the N-1 record in Chapter 6, and Sections 14.4 and 14.5 in Chapter 14). The following considerations should be kept in mind when using the Timoshenko option:
• Bending stresses and curvatures are computed correctly only at the midspan of the beam element. Bending stress resultants and moments, however, are valid at all element integration points.

• Plasticity cannot be used with the Timoshenko option, since the contribution of the shear stress cannot be correctly accounted for.

• Timoshenko beam cross-sectional properties must be defined with respect to the principal axes of the section, i.e., the product of inertia must be zero in the \((\bar{y}, \bar{z})\) system, as expressed below.

\[
I_{yz} = \int A \bar{y} \bar{z} dA = 0
\]

The sequence of J records is repeated \(NTAB\) times (B-3).

See Section 16.2 for more information about this.

<table>
<thead>
<tr>
<th>ITAB</th>
<th>KCROSS</th>
<th>MATB</th>
<th>NSUB</th>
<th>TORJ</th>
<th>SCY</th>
<th>SCZ</th>
<th>NSOYZ</th>
<th>KAPY</th>
<th>KAPZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>user-assigned cross-section number; (1 \leq ITAB \leq NTAB) (B-3)</td>
<td>cross-section type. Thermal loading is permitted (via user-written subroutine TEMP) only for (KCROSS = 2) or (3). Plasticity is permitted only for (KCROSS = 2) or (3).</td>
<td>material number, as defined by (ITAM) in the Material Table (I-1)</td>
<td>number of subelements ((NSUB \leq 10)); meaningful for subelement stiffeners ((KCROSS = 2) or (3)) only</td>
<td>torsional constant, (J), having units of (l^4), where (l) is length. The torsional stiffness for a beam of length (L) is (JG/L), where (G) is the shear modulus. For example, (J = (1/3)C \sum bh^3) for open sections composed of long, narrow rectangles. Stiffeners of shapes such as “blade”, “tee”, “I” fall into this category; but “hat” stiffeners do not, since they form a closed cross-section. In the above expression for (J), (h) is the smallest dimension for each rectangle and (C) is a correction factor, generally taken as 1.0 for stiffeners of normal proportions. (TORJ) is not used for (KCROSS = 4).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\bar{y}) coordinate of shear center</td>
<td>(\bar{z}) coordinate of shear center</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**NSOYZ**

number of stress output points per cross-section; **NSOYZ** \( \leq 4 \). **NSOYZ** is meaningful for **KCROSS** = 1 only.

**KAPY**

dimensionless transverse-shear shape factor, \( \kappa_y \), accounting for flexural shear deformation in the \((x, y)\) plane \((i.e.,\) bending about the \(z\)-axis). Set **KAPY** = 0 to omit the effects of flexural-shear deformation. This is a floating point entry.

**KAPZ**

dimensionless transverse-shear shape factor, \( \kappa_z \), accounting for flexural shear deformation in the \((x, z)\) plane \((i.e.,\) bending about the \(y\)-axis). Set **KAPZ** = 0 to omit the effects of flexural-shear deformation. This is a floating point entry.

\( \kappa_y, \kappa_z \) are shape factors, accounting for variation of transverse shear stress over the beam cross section in the \((y, z)\) directions, respectively. As an example, for parabolic variation over a rectangular cross-section, \( \kappa_y = \kappa_z = \kappa = 5/6 \).

**STAGS** will not report transverse shear stress values \((\tau_y, \tau_z)\). However, stress resultants \((V_y, V_z)\) are computed, and these are related to \( \kappa_y, \kappa_z \) by the expressions

\[
V_y = \kappa_y GA \frac{\gamma_y}{\kappa_y}, \quad V_z = \kappa_z GA \frac{\gamma_z}{\kappa_z}
\]

where \( G \) is the shear modulus, \( A \) is the cross-sectional area, and \((\frac{\gamma_y}{\kappa_y}, \frac{\gamma_z}{\kappa_z})\) are the average transverse shear strains. If a discrete value of \((\tau_y, \tau_z)\) is desired, the analyst may compute this by referring to the assumption of shear-stress variation which was used to derive \( \kappa_y, \kappa_z \). For parabolic variation in a particular coordinate direction, the maximum stress, occurring at mid-depth, is \( \tau_{\text{max}} = (3/2)\tau_{\text{avg}} \).

The cross-sectional area effective in developing transverse shear stress is sometimes assumed to be different than that effective in developing normal stress. For example, a common assumption for shapes similar to “tee”, “I”, channel, etc., is that the flanges do not develop shear stress. Thus, the “shear area” is taken as the area of the web \((A_w)\), with the familiar \( \kappa = 5/6 \), since the web is a simple rectangle. Since **STAGS** does not permit independent shear area definition, this effect can be accounted for in the specification of \( \kappa \). Thus, one might define \( \kappa = (5/6) \cdot (A_w/A) \) for a “tee” cross-section. This is left to the analyst’s judgment.

**CAUTION:** Some references define a form factor \((f)\) which is the inverse of the shape factor used by **STAGS** \((f = 1/\kappa)\). \( \kappa_y, \kappa_z \), as used in **STAGS**, must satisfy (5.23).

```plaintext
if (KCROSS = 1) then go to J-2a
elseif (KCROSS = 2) then go to J-3a
elseif (KCROSS = 3) then go to J-3b
elseif (KCROSS = 4) then go to J-4a
```
**J-2a General Cross-Section—Record 1**

Records of type J-2a/b are included only if the cross-section is of general type, $k_{CROSS} = 1$ (J-1). Record J-2a gives the section properties of a beam (stiffener) of general type. The origin of the $(\gamma, \zeta)$ cross-section coordinate system must coincide with the cross-section centroid; $(\gamma, \zeta)$ may be oriented to simplify computation of cross-sectional properties. See Figure 5.5 on page 5-122.

<table>
<thead>
<tr>
<th>BA</th>
<th>BIY</th>
<th>BIZ</th>
<th>BIYZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>cross-sectional area</td>
<td>moment of inertia about $\gamma$-axis</td>
<td>moment of inertia about $\zeta$-axis</td>
<td>product of inertia, $\int_A \gamma \zeta dA$</td>
</tr>
</tbody>
</table>

- $n_{SOY} (J-1)$ number of stress output points

if $(n_{SOY} > 0)$ then go to J-2b
else follow instructions at end of J-2b
Figure 5.5  Beam cross sections.

The beam cross-section axis, completes a right-handed \((x, y, z)\) cross-section coordinate system. The cross-section is \textit{positioned} by specifying \textbf{ECY} and \textbf{ECZ}, and it is \textit{oriented} by specifying \textbf{ZETA}.

See Figure 6.5 on page 6-52 and Figure 8.1 on page 8-12.
J-2b General Cross-Section—Record 2

This record is included only if stress output points are to be defined for the cross-section, \text{nsoyz} > 0 (J-1). Locations for stress output are defined in \((\bar{y}, \bar{z})\) cross-section coordinates.

\[
(\text{SOY}(i), \text{SOZ}(i), i = 1, \text{nsoyz})
\]

- \text{SOY}(i) \ y \text{ coordinate of stress output point } i
- \text{SOZ}(i) \ z \text{ coordinate of stress output point } i

\text{NTAB} (B-3) \ number \ of \ entries \ in \ Cross \ Section \ Table

if (\text{NTAB} \ cross \ sections \ have \ been \ defined) \ then

\text{follow \ instructions \ at \ end \ of \ J-4b}

else \ return \ to \ J-1
J-3a General Subelement Cross-Section

This record is included only if $K_{CROSS} = 2$ (J-1). It is substituted by a J-3b record if $K_{CROSS} = 3$ (subelements are rectangular). One record is used for each subelement, i.e., the record is repeated $NSUB$ times (J-1) for each cross section of this type. For subelement cross-sections of this type, stress output is always given at the centroid. See Figure 5.5 on page 5-122.

$\begin{array}{cccccccc}
SA(i) & SY(i) & SZ(i) & SIY(i) & SIZ(i) & SIYZ(i) & ISP(i) \\
\end{array}$

$SA(i)$ area of subelement

$SY(i)$ $y$ coordinate at centroid of subelement

$SZ(i)$ $z$ coordinate at centroid of subelement

$SIY(i)$ subelement moment of inertia relative to its centroid, $SZ(i)$, about a line parallel to the $y$ axis

$SIZ(i)$ subelement moment of inertia relative to its centroid $SY(i)$, about a line parallel to the $z$ axis

$SIYZ(i)$ subelement product of inertia relative to its centroid $(SY(i), SZ(i))$ for the $(y, z)$ directions

$ISP(i)$ print flag for stress output. Stress output must be activated on the R-1 record (output control) for each shell unit in which stress output is desired and on the V-1 record for each element unit; thus, stress results are printed only when specified on R-1/V-1 and on the associated J-3a.

0 − print stresses for this subelement

1 − do not print stresses for this subelement

$NSUB$ (J-1) number of subelements

$NTAB$ (B-3) number of entries in the Cross Section Table

if $(NSUB$ subelements have been defined) then
if $(NTAB$ cross sections have been defined) then

follow instructions at end of J-4b

else return to J-1

else continue defining J-3a
J-3b Rectangular Subelement Cross-Section

This record is included only if \( \text{KCROSS} = 3 \) (J-1). One record is used for each subelement, \( i.e., \) the record is repeated \( \text{NSUB} \) times (J-1) for each cross section of this type. Stress output can be obtained only at subelement corners. See Figure 5.5 on page 5-122.

\[
\begin{array}{cccccc}
Y1(i) & Y2(i) & Z1(i) & Z2(i) & ISOC(i) \\
\end{array}
\]

- \( Y1(i) \): \( \bar{y}_1 \) for subelement \( i \)
- \( Y2(i) \): \( \bar{y}_2 \) for subelement \( i \)
- \( Z1(i) \): \( \bar{z}_1 \) for subelement \( i \)
- \( Z2(i) \): \( \bar{z}_2 \) for subelement \( i \)
- \( ISOC(i) \): a 4-digit binary integer governing stress output. The four digits correspond to corners 1–4 as shown in Figure 5.5, in that order; a one indicates stress output at the corresponding corner, a zero suppresses output. For example, \( ISOC(i) = 0101 \) specifies stress output at corners 2 and 4 of subelement \( i \).

- \( \text{NSUB} \): (J-1) number of subelements
- \( \text{NTAB} \): (B-3) number of entries in the Cross Section Table

if (\( \text{NSUB} \) subelements have been defined) then
  if (\( \text{NTAB} \) cross sections have been defined) then
    follow instructions at end of J-4b
  else return to J-1
else continue defining J-3b
J-4a Arbitrary Cross-Section—Record 1

This record is included only if $\text{Kcross} = 4$ (J-1). The area input here is used to compute the mass per unit length of the beam from the density referenced by material number $\text{MatB}$ (J-1).

<table>
<thead>
<tr>
<th>BMA</th>
<th>cross-section area</th>
</tr>
</thead>
</table>

$\text{go to J-4b}$
J-4b Arbitrary Cross-Section—Record 2

This record is for input of the general material stiffness matrix for a beam cross section. The matrix relates stress resultants to reference-axis strain, curvature, and unit twist by the relation

\[
\begin{bmatrix}
N_x \\
M_z \\
M_y \\
T
\end{bmatrix} =
\begin{bmatrix}
C_{11} & C_{12} & C_{13} & C_{14} \\
C_{21} & C_{22} & C_{23} & C_{24} \\
C_{31} & C_{32} & C_{33} & C_{34} \\
C_{41} & C_{42} & C_{43} & C_{44}
\end{bmatrix}
\begin{bmatrix}
\varepsilon_x \\
\kappa_z \\
\kappa_y \\
\alpha
\end{bmatrix}
\]

\(\kappa_z, M_z\) are associated with bending about the \(z\) axis, and \(\kappa_y, M_y\) are associated with bending about the \(y\) axis. Stress resultants are related to axial and torsional-shear stresses by the relations

\[
N_x = \int_A \sigma_{xx}(y, z) \, dydz \\
M_z = \int_A y \sigma_{xx}(y, z) \, dydz \\
M_y = \int_A z \sigma_{xx}(y, z) \, dydz
\]

\[
T = \int_A \left[y \tau_{xz}(y, z) - z \tau_{xy}(y, z)\right] \, dydz
\]

Reference-axis strain, curvature, and unit twist are defined by the relations

\[
\varepsilon_x = u_x, \quad \kappa_z = -v_{xx}, \quad \kappa_y = -w_{xx}, \quad \alpha = (R_x^2 - R_z^1) / l
\]

where \(l\) is the length of the beam, \(R_1^1\) and \(R_2^2\) are the axial rotations at nodes 1 and 2, and the unit twist \((\alpha)\) has units of radians/length. In all of the above equations, \((x, y, z)\) represents the \((x, y, z)\) cross-section coordinate system, and \((u, v, w)\) are displacements in the \((x, y, z)\) directions (see Figure 6.5 on page 6-52 and Figure 8.1 on page 8-12).

Note that the full matrix is input, even though it must be symmetric.
(CCC(i,j), j=1,4), i=1,4

\(\text{CCC}(i,j)\) \hspace{1em} \text{element} \ c_{ij} \text{ in the cross-section stiffness matrix}

**NOTE:** \hspace{1em} 16 entries on one record, input in row-major order.

- **NTAB** (B-3) \hspace{1em} number of entries in the Cross Section Table
- **NTAW** (B-3) \hspace{1em} number of entries in the Wall Fabrication Table
- **NTAP** (B-3) \hspace{1em} User Parameters flag
- **NUNITS** (B-2) \hspace{1em} number of shell units

if (NTAB cross sections have been defined) then
  if (NTAW > 0) then \hspace{1em} go to \ K-1
  elsif (NTAP > 0) then \hspace{1em} go to \ L-1
  elseif (NUNITS > 0) then \hspace{1em} go to \ M-1
  else \hspace{1em} follow instructions at end of \ R-3
else \hspace{1em} return to \ J-1
K-1 Shell Wall Properties

The records K-1–K-6 define various shell wall configurations. They are included only if $\text{NTAW} > 0$ (B-3).

Data in the Wall Fabrication Table are defined either by input on regular data records or by a user-written subroutine WALL. A user-written subroutine must be used if

- The shell wall construction is not included among the standard types defined below.
- Properties or dimensions vary with the surface coordinates.
- Material properties vary continuously (not stepwise at interface between layers) through the shell wall thickness.

Shell wall properties are defined in $(\xi, \eta)$ fabrication coordinates (see Section 4.1).* These coordinates are chosen so that definition of material properties is as easy as possible. They need not coincide with shell or element coordinates.

The shell wall constructions that can be defined by use of data records are:

1) Shells consisting of a number of orthotropic layers.
2) Shells consisting of fiberwound layers.
3) Shells reinforced by a corrugated skin.
4) Properties defined by use of the elements of the shell wall stiffness matrix.
5) Any of the above with smeared stiffeners. These stiffeners need not be along coordinate lines, but the angle between the coordinate directions and the stiffener attachment line may not vary along the stiffener.

A shell with a homogenous isotropic or orthotropic wall is defined by use of option (1).

Fiberwound layered shells may be defined by use of option (2). The shell wall stiffness parameters are then computed from the basic properties of the constituent materials. However, no method by which such parameters are computed has been generally accepted. If the stiffness parameters for the individual layers are known (from test, for example), better results may be obtained by use of option (1). The computation of the stiffness matrix from constituent material

* Note the similarity to beam cross-section coordinates, in which the $\bar{x}$ coordinate defines the beam axis, with the $(\eta, z)$ coordinates lying in the cross-section. For shells, the $(\xi, \eta)$ coordinates lie in the shell surface, with $z$ defining the shell normal.
data should be used only if the individual layer properties required for option (1) cannot be obtained.

Material properties in each layer are defined in material coordinates (see Section 4.1), which are established by rotating the (, ) fabrication coordinates through an angle , a right-handed rotation about the axis. and are independently defined for each layer.

For a shell wall consisting of only a corrugated skin, use option (3).

If stiffeners are defined as “smeared,” their contribution to the shell wall stiffness is “smeared out” uniformly over the shell surface. As a consequence they cannot be used if temperatures vary within the structure.

For an “isogrid” or a skin with skew stiffeners, or for any wall whose stiffness properties have been computed by other means, use option (4).

Properties of smeared stiffeners can be defined by reference to the Cross Section Table. If stiffener spacing, the cross-section properties, or the angle with respect to the coordinate lines
varies with the shell coordinates, the data must be defined in a user-written subroutine WALL. Stiffener data in the Cross Section Table can be referenced in WALL.

Some of the data defined on this record refer to layered shells (\texttt{KWALL} = 1 or 2). For walls with corrugations, \texttt{NLAY}, \texttt{NLIP} are irrelevant. The \( \phi_1 \) direction for corrugated shells coincides with the axis of the corrugations.

The sequence of K records is repeated \texttt{NTAW} times (B-3).

See 16.1 “Shell Results” on page 16-1 for more information.

\begin{verbatim}
ITAW  KWALL  NLAY  NLIP  NSMRS  SHEAR1  SHEAR2
\end{verbatim}

\begin{description}
\item ITAW \user-assigned wall configuration number; \texttt{ITAW} \( \leq \texttt{NTAW} \) (B-3)
\item KWALL \wall construction type
\begin{itemize}
\item 1 – general layered
\item 2 – fiberwound — INACTIVE
\item 3 – corrugated
\item 4 – general; elements of “stiffness matrix” defined
\end{itemize}
\item NLAY \number of layers through the thickness (\texttt{NLAY} \( \leq 100 \));
relevant for layered walls (\texttt{KWALL} = 1 or 2) only
\item NLIP \number of integration points through the thickness of each individual layer to be used for plasticity and thermal effects. Setting \texttt{NLIP} = 0 or 1 implies that all properties, including temperature, are considered uniform within each layer. For plasticity, \texttt{NLIP} must be odd and 0 \( \leq \texttt{NLIP} \leq 9 \). See Table 5.4.
\end{description}

\begin{center}
\textbf{Table 5.4} \quad \textbf{Suggested values for \texttt{NLIP}.}
\end{center}

\begin{tabular}{|c|c|c|}
\hline
\texttt{NLIP} & material behavior & thermal loading \\
\hline
1 & linear elasticity & uniform temperature \\
3 & & linear gradient \\
5 & plasticity & quadratic gradient \\
7 & severe plasticity & \\
9 & & \\
\hline
\end{tabular}
**NSMRS** number of sets of smeared stiffeners \((\text{NSMRS} \leq 3)\); material behavior is restricted to linear elasticity in walls containing smeared stiffeners

**SHEAR1** shear correction factor (see note, below)

**SHEAR2** shear correction factor (see note, below)

**Note:** The default value for each of these shear correction factors is \(5/6\)—which implies a parabolic distribution through the thickness (exact for homogeneous, isotropic prismatic fabrications). Setting either factor to unity implies a uniform distribution, which is unconservative.

```
if (KWALL = 1) then go to K-2
elseif (KWALL = 2) then go to K-3a
elseif (KWALL = 3) then go to K-4a
elseif (KWALL = 4) then go to K-5a
```
**K-2 Layered Wall**

The K-2 record defines a layered shell wall. It is included only if \( KWALL = 1 \) (K-1) and is repeated \( NLAY \) times. Records to be defined later will control whether stresses are printed in any given shell unit or part of the element unit. The layers are stacked in the order read and in the direction of increasing \( \zeta \). The origin of the \((x, y)\) system is located at the middle surface of the shell (see Figure 5.6 on page 5-130). Eccentricity with respect to the reference surface is defined on the M-5 record. The material properties for each layer are given through reference to the Material Table (I-1).

---

**MATL TL ZETL LSOL**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATL</td>
<td>layer material number; select from the Material Table</td>
</tr>
<tr>
<td>TL</td>
<td>layer thickness</td>
</tr>
<tr>
<td>ZETL</td>
<td>material (e.g., fiber) orientation angle, in degrees; a right-handed rotation about ( \zeta ). ( ZETL ) is the angle ( \zeta ) between ( \phi_1 ) and ( \bar{x} ).</td>
</tr>
<tr>
<td>LSOL</td>
<td>layer stress output control:</td>
</tr>
<tr>
<td>0</td>
<td>do not print stresses this layer</td>
</tr>
<tr>
<td>1</td>
<td>print stresses this layer; applicable only at locations where overall stress output is required</td>
</tr>
<tr>
<td>2</td>
<td>print stresses and strains at all ( NLIP ) (K-1) integration points through the thickness; applicable only in plastic analysis</td>
</tr>
</tbody>
</table>

![Figure 5.7](image)

**Figure 5.7** Coordinate systems for walls.

- \( \phi_2 \)
- \( \zeta = ZETAL \) layered wall
- \( \zeta = ZETW \) fiber-reinforced wall (INACTIVE)

---

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLAY</td>
<td>( (K-1) ) number of layers</td>
</tr>
<tr>
<td>NSMRS</td>
<td>( (K-1) ) number of sets of smeared stiffeners</td>
</tr>
</tbody>
</table>

if \( (NLAY \) layers have been defined) then
if \( (NSMRS > 0) \) then \( go \ to \ K-6 \)
else \( follow \ instructions \ at \ end \ of \ K-6 \)
else \( continue \ defining \ K-2 \)
K-3a Fiber Reinforced Wall—Record 1

INACTIVE

Records K-3a/b define a fiber reinforced wall by use of the properties of the constituent materials. Material properties for the matrix and fibers are obtained from the Material Table (I-1). Usually, better results are obtained if the layer properties can be defined directly on K-2 records.

The K-3a/b records are included only if Kwall = 2 (K-1).

<table>
<thead>
<tr>
<th>MATF</th>
<th>MATM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATF</td>
<td>material number (I-1) for fibers</td>
</tr>
<tr>
<td>MATM</td>
<td>material number (I-1) for matrix</td>
</tr>
</tbody>
</table>

*go to K-3b*
The layers are numbered in the direction of increasing value of $z$. The $\phi_1$ direction for the layer coincides with the winding angle.

One K-3b record is read for each layer, *i.e.*, the record is repeated $\text{NLAY}$ times (K-1).

<table>
<thead>
<tr>
<th>TT</th>
<th>layer thickness</th>
</tr>
</thead>
<tbody>
<tr>
<td>XX</td>
<td>matrix content (by volume)</td>
</tr>
<tr>
<td>ZETW</td>
<td>winding angle in degrees</td>
</tr>
<tr>
<td>O</td>
<td>contiguity factor</td>
</tr>
</tbody>
</table>

if ($\text{NLAY}$ layers have been defined) then
  if ($\text{NSMRS} > 0$) then go to K-6
  else follow instructions at end of K-6
else continue defining K-3b
K-4a Corrugation Stiffened Wall—Record 1

The records K-4a/b define a corrugation stiffened wall, $KWALL = 3$ (K-1).

Figure 5.8  Corrugation-stiffened shell wall.

---

<table>
<thead>
<tr>
<th>MATC</th>
<th>MATS</th>
<th>CT</th>
<th>CC</th>
<th>CH</th>
<th>CD</th>
<th>CB</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATC</td>
<td>material number (I-1) for corrugated sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MATS</td>
<td>material number (I-1) for skin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CT</td>
<td>thickness of corrugated sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CH</td>
<td>See Figure 5.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CB</td>
<td>centerline-to-centerline spacing of corrugations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$go\ to\ K-4b$
# K-4b Corrugation Stiffened Wall—Record 2

<table>
<thead>
<tr>
<th>TS</th>
<th>PHI</th>
<th>ANC</th>
</tr>
</thead>
<tbody>
<tr>
<td>thickness of smooth skin</td>
<td>reduction factor for torsional stiffness</td>
<td>corrugation location</td>
</tr>
<tr>
<td>0</td>
<td>–</td>
<td>inside corrugation</td>
</tr>
<tr>
<td>1</td>
<td>–</td>
<td>outside corrugation</td>
</tr>
</tbody>
</table>

NSMRS (K-1) number of sets of smeared stiffeners

if (NSMRS > 0) then *go to K-6*
else *follow instructions at end of K-6*
**K-5a General Wall—Record 1**

This record is included only if $\text{KWALL} = 4$ (K-1). The thickness needs to be defined to permit mass matrix computation and for thermal analysis ($\text{ITEMP}, \text{C-1}$). Mass per unit area is $\text{TA} \cdot \text{RHO}$, where $\text{RHO}$ is the weight density given on the I-2 record. The I-2 record is included so that density and thermal expansion coefficients can be defined; the other entries are irrelevant.

<table>
<thead>
<tr>
<th>TA MAT ITVS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA</td>
</tr>
<tr>
<td>MAT</td>
</tr>
<tr>
<td>ITVS</td>
</tr>
</tbody>
</table>

- 0 – ignore transverse shear
- 1 – include transverse shear coefficients (K-5c)

$\text{ITVS}$ must be set to 1 if this wall is to be assigned to an element which is transverse-shear deformable (currently, only the $\text{E330}$ triangular and the $\text{E480}$ quadrilateral shell elements). Otherwise, the element stiffness matrix will be singular. For elements which do not permit transverse-shear deformation, $\text{ITVS}$ is irrelevant.

*go to K-5b*
K-5b General Wall—Record 2

This record is for input of the general material stiffness matrix. This matrix relates stress resultants to reference-surface strains and curvatures by the relation

\[
\begin{bmatrix}
N_{xx} \\
N_{yy} \\
N_{xy} \\
M_{xx} \\
M_{yy} \\
M_{xy}
\end{bmatrix} =
\begin{bmatrix}
C_{11} & C_{12} & C_{13} & C_{14} & C_{15} & C_{16} \\
C_{21} & C_{22} & C_{23} & C_{24} & C_{25} & C_{26} \\
C_{31} & C_{32} & C_{33} & C_{34} & C_{35} & C_{36} \\
C_{41} & C_{42} & C_{43} & C_{44} & C_{45} & C_{46} \\
C_{51} & C_{52} & C_{53} & C_{54} & C_{55} & C_{56} \\
C_{61} & C_{62} & C_{63} & C_{64} & C_{65} & C_{66}
\end{bmatrix}
\begin{bmatrix}
\varepsilon_{xx} \\
\varepsilon_{yy} \\
\gamma_{xy} \\
\kappa_{xx} \\
\kappa_{yy} \\
\kappa_{xy}
\end{bmatrix}
\]

where \((x, y, z)\) represent \((\tilde{x}, \tilde{y}, \tilde{z})\) fabrication coordinates.

**NOTE:** Engineering strains are used: \(\gamma_{xy} = \varepsilon_{xy} + \varepsilon_{yx}\) and \(\kappa_{xy} = -2w_{xy}\)

The full matrix is input, even though it must be symmetric.

\[
(\text{CCC}(i,j), \ j = 1, 6 \ ), \ i = 1, 6
\]

**CCC(i,j)** element \(C_{ij}\) in the shell wall stiffness matrix

**Note:** 36 entries on one record, input in row-major order.

- **ITVS** (K-5a) transverse shear flag
- **NSMRS** (K-1) number of sets of smeared stiffeners

if **ITVS** > 0 then  \textit{go to} K-5c
elseif **NSMRS** > 0 then \textit{go to} K-6
else \textit{follow instructions at end of} K-6
**K-5c General Wall—Record 3**

Record K-5c is used to input the two-by-two matrix of transverse shear coefficients used only for transverse shear-deformable elements such as the **E480** quadrilateral shell (see Section 14.5). This record is read only if **ITVS > 0** (K-5a).

This matrix relates transverse shear stress resultants to transverse shear strains by the relation below, where the bar over \( \gamma \) indicates average through-thickness values.

\[
\begin{bmatrix}
Q_x \\
Q_y
\end{bmatrix}
= \begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix}
\begin{bmatrix}
\bar{\gamma}_{zx} \\
\bar{\gamma}_{yz}
\end{bmatrix}
\]

**Note:** Engineering strains are used: \( \tilde{\gamma}_{zx} = \varepsilon_{xz} + \varepsilon_{zx} \) and \( \tilde{\gamma}_{yz} = \varepsilon_{yz} + \varepsilon_{zy} \)

The full matrix is input, even though it must be symmetric.

\( (CTS(i,j), \ j = 1, 2 \), \ i = 1, 2 \)

<table>
<thead>
<tr>
<th>CTS(i,j)</th>
<th>transverse shear element ( C_{ij} ) in the shell wall stiffness matrix</th>
</tr>
</thead>
</table>

**NOTE:** 4 entries on one record, input in row-major order.

- **NSMRS** \( (K-1) \) number of sets of smeared stiffeners
  
  if **(NSMRS > 0)** then *go to K-6*
  
  else *follow instructions at end of K-6*
**K-6 Smeared Stiffener**

This record defines the properties of smeared stiffeners by referring to the Cross Section Table. The record is not included if $\text{NSMRS} = 0$ (K-1).

The record is repeated $\text{NSMRS}$ times (K-1).

<table>
<thead>
<tr>
<th>ICROSM</th>
<th>SPASM</th>
<th>ZETSM</th>
<th>XSISM</th>
<th>ECZSM</th>
</tr>
</thead>
</table>

**ICROSM**

Cross-section number, as defined by ITAB (J-1) in the Cross Section Table; ICROSM = 0 indicates that the stiffener is defined in user-written CROSS.

**SPASM**

Stiffener spacing; see the following discussion regarding units.

If SPASM $> 0$, then its units are taken to be length.

If SPASM $< 0$, then its units are interpreted according to the following. When ZETSM = 0° (smeared stringers) or ZETSM = 90° (smeared rings), SPASM is input in the same units as the Y surface coordinate (stringer) or the X surface coordinate (ring). For smeared skew stiffeners (0° $< |\text{ZETSM}| < 90°$) SPASM is interpreted according to

$$
\Delta S = \alpha_1 \Delta \theta \frac{1}{\sqrt{\frac{1}{\sin^2 \text{ZETSM}} + \left(\frac{\alpha_1}{\alpha_2}\right)^2 \cos^2 \text{ZETSM}}} 
$$

where $dS_1 = \alpha_1 dX$, $dS_2 = \alpha_2 dY$, and $\Delta \theta = |\text{SPASM}|$. It can be seen in the above equation that when ZETSM = 0° or 90°, SPASM is expressed in the same units as Y or X, respectively.

**ZETSM**

Angle $\zeta$ between the $\bar{x}$ axis and the $x_w$ axis (see Figure 5.9 on page 5-142)

**XSISM**

Angle $\zeta$ between the $z$ axis and the $Z'$ axis (see Figure 6.5 on page 6-52)

**ECZSM**

Eccentricity of the smeared stiffener with respect to the shell reference surface, i.e., the value of the shell coordinate $Z'$ at the origin of the $(\gamma, z)$ cross-section coordinate system (see Figure 6.5 on page 6-52)
Figure 5.9  Orientation of smeared stiffener.

\[ \zeta = \text{ZETSM} \quad \text{smeared stiffener} \]

**NSMRS (K-1)**  number of sets of smeared stiffeners

**NTAW (B-3)**  number of entries in the Wall Fabrication Table

**NTAP (B-3)**  User Parameters flag

**NUNITS (B-2)**  number of shell units

if (NSMRS K-6 records have been defined) then
  if (NTAW wall fabrications have been defined) then
    if (NTAP > 0) then  go to  L-1
    elseif (NUNITS > 0) then  go to  M-1
    else  follow instructions at end of  R-3
  else  return to  K-1
else  continue defining  K-6
L-1 User Parameters Summary

The L-1/L-2a/L-2b series is used to define constants for access in user-written subroutines. This L record series is included only if \( \text{NTAP} > 0 \) (B-3).

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NPI</td>
<td>number of integer parameters to be read on L-2a; ( \text{NPI} \leq 200 )</td>
</tr>
<tr>
<td>NPF</td>
<td>number of floating-point parameters to be read on L-2b; ( \text{NPF} \leq 200 )</td>
</tr>
</tbody>
</table>

- if \( \text{NPI} > 0 \) then go to L-2a
- elseif \( \text{NPF} > 0 \) then go to L-2b
- else follow instructions at end of L-2b
**L-2a Integer Parameters**

This record is used to read \textbf{NPI} integers (L-1). These integer parameters may then be accessed in any user-written subroutine via the FORTRAN type and labeled common statements:

\[
\text{INTEGER UserInt}\\
\text{COMMON /UPI/ UserInt(200)}
\]

\textbf{USERINT}(i), \textit{i} = 1, \textbf{NPI}

\textbf{USERINT}(i) integer constant

\textbf{NPF} (L-1) number of floating-point parameters

if \textbf{NPF} > 0 then \textit{go to L-2b} \\
else \textit{follow instructions at end of L-2b}
**L-2b Floating-Point Parameters**

This record is used to read NPF floating-point constants (L-1). These floating-point parameters may then be accessed within any user-written subroutine via the FORTRAN type and labeled common statements:

```
REAL UserFlo
COMMON /UPF/ UserFlo(200)
```

**USERFLO(i), i = 1, NPF**

USERFLO(i)  floating-point constant

- **NUNITS (B-2)**  number of shell units

  if (NUNITS > 0) then  go to M-1
  else  follow instructions at end of R-3
5.7 References for Chapter 5


Model Input—Shell Units

Shell units are numbered sequentially, in the order in which they are defined, from 1 to \( \text{NUNITS} \), where \( \text{NUNITS} \) is the number of shell units (B-2).

The group of records in the \( \text{M–R} \) series describes the shell units. They are defined in serial fashion, completely describing one shell unit before proceeding to the next. When \( \text{NUNITS} \geq 2 \), the \( \text{M–R} \) series is defined for the first shell unit, the series is repeated for the second shell unit, and so forth, until all of the \( \text{NUNITS} \) shell units have been defined.

The \( \text{NUNITE} \) (B-2) element units are then defined in a similar manner using the \( \text{S–V} \) series (see Chapter 7 “Model Input—Element Units (1)”).

The \( \text{M–R} \) records contain information as summarized below:

- \( \text{M} \) records: Geometry
- \( \text{N} \) records: Discretization
- \( \text{O} \) records: Discrete Stiffeners
- \( \text{P} \) records: Boundary Conditions
- \( \text{Q} \) records: Specified Loadings and/or Displacements
- \( \text{R} \) records: Output Control

A STAGS model may contain any number of shell units, combined with any number of element units. Restrictions in previous versions of STAGS on the allowable number of units have been removed.
6.1 Geometry

Standard Shell Surfaces

Definition of standard shell surfaces requires particular awareness of four coordinate systems (see Section 4.1 “Coordinate Systems” on page 4-1):

- \((x, y, z)\) branch coordinates
- \((X, Y)\) surface coordinates
- \((X', Y', Z')\) shell coordinates
- \((x_g, y_g, z_g)\) global coordinates

The standard shell surfaces, \(\text{ISHELL} = 2-12\) (M-1), are shown in the following figures. Each figure contains a sketch of a particular shell type, along with labeled coordinate axes for both branch coordinates and surface coordinates. Equations to the right of each figure define user input (\(\text{PROPS}, \text{M-2}\)) in terms of these two coordinate systems. Furthermore, expressions relating branch coordinates to surface coordinates are given beneath each sketch. While the user generally need not be concerned with these surface-to-branch relationships, they are definitive for resolving any ambiguity.

Shell coordinates form an orthogonal set, without exception, and at each shell-unit node, the computational degrees-of-freedom (dof) are determined by this system. That is to say, \((x'', y'', z'')\) is coincident with \((X', Y', Z')\) at each node in a shell unit. \((X, Y)\) also form an orthogonal set, \((X', Y')\) being coincident, with the single exception of the Elliptic Cone (\(\text{ISHELL} = 9\)). On this type of surface, the meridians and parallels do not cross at right angles, except for the three degenerate surfaces (elliptic cylinder, circular cone, circular cylinder). However, the \((X', Y', Z')\) system is still orthogonal in this case. The circumferential \((Y)\) and shell-normal directions are chosen as the \((Y', Z')\) directions, and \(Y' \times Z' = X'\) establishes the orthogonal system.

After defining a shell unit (branch) via \(\text{ISHELL/PROPS}\), it must be positioned in global space via \(\text{IGLOBE}\) (M-1). Globally positioning a shell branch is logically equivalent to defining a branch-to-global transformation, that is specifying where the origin of the branch coordinate system is located, and how it is oriented (rotated) to position the shell branch properly. \textit{STAGS} provides tremendous flexibility and automation for globally positioning shell branches, so the user need not be concerned with mathematical details. Still, it is insightful to bear in mind that the effect
of IGLOBE selection is to define a branch-to-global transformation. For example, in the simplest case, IGLOBE = 0, the branch system is coincident with the global system, thus defining the identity transformation.

RECTANGULAR PLATE

\[
\begin{array}{cccc}
1 & 2 \\
4 & 3 \\
\end{array}
\]

\[
x = X \quad y = Y \quad z = 0
\]

\[
\begin{align*}
PROP(1) &= X_1 \\
PROP(2) &= X_4 \\
PROP(3) &= Y_1 \\
PROP(4) &= Y_2
\end{align*}
\]

ISHELL = 2
**QUADRILATERAL PLATE**

\[ f_1 = (1 - \xi)(1 - \eta) \quad f_2 = (1 - \xi)\eta \quad f_3 = \xi\eta \quad f_4 = \xi(1 - \eta) \]

\[
\begin{align*}
x &= f(X, Y) = \sum_{i=1}^{4} f_i \cdot X_i \\
y &= g(X, Y) = \sum_{i=1}^{4} f_i \cdot Y_i \\
z &= 0
\end{align*}
\]

**NOTE:** (X, Y) = (x, y); i.e., the surface coordinate directions are parallel to the branch coordinate directions.

A quadrilateral plate with (X, Y) derived from the (\(\xi, \eta\)) lines can be generated with a user-defined shell surface (ISHELL = 1). For an example, see “LAME Reference Surface Geometry” on page 12-19.
ANNULAR PLATE

\[ ISHELL = 4 \]

\[ R_a \]
\[ R_b \]
\[ \theta_a \]
\[ \theta_b \]
\[ Y \]
\[ X \]
\[ Z \]
\[ X_1 = R_a \]
\[ X_4 = R_b \]
\[ Y_1 = \theta_a \]
\[ Y_2 = \theta_b \]

\[ X^2 = y^2 + z^2 = r^2 \]

\[ x = 0 \quad y = g(X, Y) = X \sin Y \quad z = h(X, Y) = X \cos Y \]

CYLINDER

\[ ISHELL = 5 \]

\[ R \]
\[ L \]
\[ Z \]
\[ X \]
\[ Y \]
\[ X_1 = x_1 \]
\[ x_4 = x_4 \]
\[ Y_1 = \theta_a \]
\[ Y_2 = \theta_b \]
\[ R \]

\[ L = X_4 - X_1 \quad y^2 + z^2 = R^2 \]

\[ x = X \quad y = g(X, Y) = R \sin Y \quad z = h(X, Y) = R \cos Y \]
CONE

\[ \text{ISHELL} = 6 \]

\[ \begin{align*}
PROP(1) &= x_1 \\
PROP(2) &= x_4 \\
PROP(3) &= Y_1 = \theta_a \\
PROP(4) &= Y_2 = \theta_b \\
PROP(5) &= R_a \\
PROP(6) &= R_b \\
\end{align*} \]

\[ R_a > R_b \text{ OK} \]

\[ L = x_4 - x_1 \]

\[ \tan \varphi = \frac{(R_b - R_a)}{L} \]

\[ r = R_a + (x - x_1) \tan \varphi \]

\[ y^2 + z^2 = r^2 \quad y = g(x, Y) = rsinY \quad z = h(x, Y) = r\cos Y \]

SPHERE

\[ \text{ISHELL} = 7 \]

\[ \begin{align*}
PROP(1) &= X_1 = \psi_a \\
PROP(2) &= X_4 = \psi_b \\
PROP(3) &= Y_1 = \theta_a \\
PROP(4) &= Y_2 = \theta_b \\
PROP(5) &= R \\
\end{align*} \]

\[ \begin{align*}
x^2 + y^2 + z^2 &= R^2 \\
x &= f(X, Y) = -R\cos X \\
y &= g(X, Y) = R\sin X \sin Y \\
z &= h(X, Y) = R\sin X \cos Y \\
\end{align*} \]
TORUS

\[ \mathbf{R} = R_a + R_b \sin X \]
\[ x = f(X, Y) = -R_b \cos X \]
\[ y = g(X, Y) = \mathbf{R} \cdot \sin Y \]
\[ z = h(X, Y) = \mathbf{R} \cdot \cos Y \]

PROP(1) = \( X_1 = \psi_a \)
PROP(2) = \( X_4 = \psi_b \)
PROP(3) = \( Y_1 = \theta_a \)
PROP(4) = \( Y_2 = \theta_b \)
PROP(5) = \( R_a \)
PROP(6) = \( R_b \)
ELLIPITIC CONE/CYLINDER

**ISHELL = 9**

\[ \text{PROP(1)} = x_1 \]
\[ \text{PROP(2)} = x_4 \]
\[ \text{PROP(3)} = Y_1 = \theta_a \]
\[ \text{PROP(4)} = Y_2 = \theta_b \]
\[ \text{PROP(5)} = R_{za} \]
\[ \text{PROP(6)} = R_{ya} \]
\[ \text{PROP(7)} = R_{zb} \]

\[ R_{za} > R_{zb} \implies \text{OK} \]
\[ L = x_4 - x_1 \]
\[ \tan \phi = (R_{zb} - R_{za})/L \]

\[ R_{za}, R_{ya}, R_{zb} \implies \text{elliptic cone} \]
\[ R_{za} = R_{zb} = R_z ; R_{ya} = R_y \implies \text{elliptic cylinder} \]
\[ R_{za} = R_{ya} = R_a ; R_{zb} = R_b \implies \text{circular cone} \quad (\text{ISHELL} = 6) \]
\[ R_{za} = R_{ya} = R_{zb} = R \implies \text{circular cylinder} \quad (\text{ISHELL} = 5) \]

\[ R_{za} < R_{zb} : X_{ap} = x - x_1 + (R_{za}/\tan \phi) \]
\[ R_{za} \geq R_{zb} : X_{ap} = x_4 - x + (R_{zb}/\tan \phi) \]

\[ r_y = (R_{ya}/R_{za})X_{ap}\tan \phi \quad r_z = X_{ap}\tan \phi \]

\[ \frac{r_y^2 + r_z^2}{r_y^2 + r_z^2} = 1 \]

\[ y = g(x, Y) = r_y \sin Y \quad z = h(x, Y) = r_z \cos Y \]

**NOTE:** \((x, Y)\) are non-orthogonal for the **elliptic cone** only; see page 6-2.
**PARABOLOID**

**ISHELL = 10**

\[
y^2 + z^2 = r^2 = 4R_a(x + R_b) \quad \overline{R} = 2\sqrt{R_a(x + R_b)}
\]

\[
y = g(x, Y) = \overline{R} \cdot \sin Y \quad z = h(x, Y) = \overline{R} \cdot \cos Y
\]

PROPs:
- PROP(1) = \(x_1\)
- PROP(2) = \(x_4\)
- PROP(3) = \(Y_1 = \theta_a\)
- PROP(4) = \(Y_2 = \theta_b\)
- PROP(5) = \(R_a\)
- PROP(6) = \(R_b\)
ELLIPSOID

\[ \frac{y^2 + z^2}{R_{yz}^2} + \frac{x^2}{R_x^2} = 1 \]

\[ X^e = \text{atan}\left(\frac{R_x\tan X}{R_{yz}}\right) \quad x = f(X, Y) = -R_x\cos X^e \]

\[ y = g(X, Y) = R_{yz}\sin X^e\sin Y \quad z = h(X, Y) = R_{yz}\sin X^e\cos Y \]
HYPERBOLOID

\[ ISHELL = 12 \]

\[ \frac{y^2 + z^2}{R_a^2} - \frac{(x + R_c)^2}{R_b^2} = 1 \]

\[ R = \frac{R_a}{R_b} \sqrt{R_b^2 + (x - x_1 + R_c)^2} \]

\[ y = g(x, Y) = \bar{R} \cdot \sin Y \]

\[ z = h(x, Y) = \bar{R} \cdot \cos Y \]
M-1 Shell Type

Six basic parameters are required by STAGS to define the geometry of a shell unit. The first of these (ISHELL) is used to specify the type of shell surface that is to be generated; the second (IGLOBE) can be used to identify the method by which the shell unit is positioned within the \((x_g, y_g, z_g)\) global coordinate system; the third and fourth (NROWS & NCOLS) are reserved for future use for specifying how the shell surface is to be tessellated (into an I-J grid) for constructing the finite elements required for the model; the fifth (NLAYS) is used to specify the number of elements to be generated through the thickness of a solid shell unit; and the sixth (NFABS) is used to specify the number of fabrications that are required for the elements to be generated.

The twelve choices that are available for the ISHELL parameter in the current version of STAGS are described below. The NROWS and NCOLS parameters on M-1 are currently ignored by the program, which uses information from the F-1 record to form the I-J nodal grid(s) required for the shell unit: the user should set these two parameters equal to zero (or omit them entirely if NLAYS and NFABS are also omitted or zero (see the discussion below). With these things in mind, the remainder of this preamble concentrates on the IGLOBE, NLAYS and NFABS parameters.

STAGS interprets the specification that IGLOBE = 0 to mean that the \((x, y, z)\) branch coordinate system for the shell unit coincides with the \((x_g, y_g, z_g)\) global coordinate system, and that no positioning transformation is required. (Please refer to “Standard Shell Surfaces” on page 6-2 & Section 4.1 “Coordinate Systems” on page 4-1.) If IGLOBE = 1 the program automatically selects 3 points on the boundary line that is common with previously defined shell units and uses these data for location of the unit. (To find out how units are joined together, see records G-1 and G-2.) If only one side of the unit is connected to a previously defined unit and that side is a straight line or has only two points, additional information is needed, and IGLOBE = 1 will not work. The IGLOBE = 2 option was originally intended for this case and is retained for compatibility, but its use is not recommended since newer, more robust, options have made it obsolete. Full generality is provided by the addition of options IGLOBE = 3, 4 and 5. With IGLOBE = 3, the M-4a record is used to define the global coordinates for the corner points 1, 2, and 3 on the shell unit. (See Figure 6.1 on page 6-19 for corner-numbering conventions.) The options IGLOBE = 4 and 5 give the user the opportunity to specify both a translation and an Euler rotation for either the origin of the branch coordinate system or corner point 1. These options will be needed for closed shell units such as complete cylinders and annular plates, where the three corner points all lie along a line; in that case, the IGLOBE = 3 option cannot be used.
The `NLAYS` parameter was not required in previous versions of STAGS because shell units there could only be created with one-dimensional beam and/or two-dimensional thin-shell elements. The current version of STAGS uses the `NLAYS` parameter to determine whether or not solid-type elements are used for the current shell unit and (if so) how many solid elements are to be generated in the thickness direction at any given location. Specification that `NLAYS = 0` (or omission of the `NLAYS` specification) instructs STAGS to generate a single layer of one-dimensional and/or thin-shell elements through the thickness of the shell unit. Under these circumstances, the `NFABS` parameter should be set equal to 0 or 1 so that STAGS will read a single M-5 record identifying the fabrication to be used.

Specification that `NLAYS > 0` instructs STAGS to generate `NLAYS` layers of solid elements through the thickness of the shell at each location. Under these circumstances, the `NFABS` parameter must be set equal to 0, to 1, to `NLAYS`, or to `(NLAYS + 2)`—depending on the type of element to be used for the shell unit (see the N-1 record, on 6-34).

When type `E830`, `E840` or `E849` (sandwich) elements are used, the `NLAYS` parameter must be set equal to the number of `core` elements to be used through the thickness of the shell, and the `NFABS` parameter must be set equal to `NLAYS + 2` or `NLAYS + 3`. With single-core-section `E830`, `E840` or `E849` elements, three fabrications are required: the first of these is used for the lower face sheet element, the second is used for the core element, and the third is used for the upper face sheet element of the `E830`, `E840` or `E849` assembly at each location on the shell unit. With multiple-core-section `E830`, `E840` or `E849` sandwiches, the `NLAYS` parameter specifies the number of `core components` to be used for each `E830`, `E840` or `E849` element, and `NFABS` must be set equal to `NLAYS + 3`. In this case, the first of these `NFABS` fabrications (identified via the `NFABS` M-5 records following M-1 `et al.`) is used for the lower face sheet, the second is used for the lowest core element, the third is used for the next core element, ... and so on, and the next-to-last fabrication is used for the upper face sheet of the `E830`, `E840` or `E849` assembly at each location on the shell. STAGS uses the last `(NFABS`th) fabrication to construct a “fictitious” `E330`, `E410` or `E480` element at each of the interfaces between successive layers of the `NLAYS` core-section components. It is recommended that each layer of elements have a fabrication record defined.

When `E881`, `E882`, `E883` or `E885` elements are used, `NFABS` may be 0, 1 or `NLAYS`: if `NFABS = 0` or 1, STAGS will read a single M-5 record that identifies the fabrication to be used for all of those elements; if `NFABS = NLAYS`, STAGS will read an M-5 record for each layer of elements—starting with the bottom layer (defined by the `ISHELL` parameter and the information on the record(s) following the current M-1 record) and continuing through and including the topmost layer of the construction.
In any event, the sequence of records M-1 through R-3 is repeated NUNITS times (B-2).

---

**ISHELL IGLOBE NROWS NCOLS NLAYS NFABS**

**ISHELL** indicates basic shell geometry; see “Standard Shell Surfaces” on page 6-2:
- 1 – defined by user-written subroutine LAME
- 2 – rectangular plate
- 3 – quadrilateral plate
- 4 – annular plate
- 5 – cylinder
- 6 – cone
- 7 – sphere
- 8 – torus
- 9 – elliptic cone/cylinder
- 10 – paraboloid
- 11 – ellipsoid
- 12 – hyperboloid

**IGLOBE** global positioning option
- 0 – the \((x, y, z)\) branch coordinate system for the shell unit coincides with the \((x_g, y_g, z_g)\) global coordinate system
- 1 – the shell unit is positioned automatically by use of points on a common boundary line; do not use if the connected side is straight or has only two grid points
- 2 – OBSOLETE; retained for compatibility, but use is not recommended due to the addition of the more general, more robust options below.
- 3 – the shell unit is positioned by specifying the global coordinates of three corner points
- 4 – the shell unit is positioned by positioning the origin of the \((x, y, z)\) branch coordinate system at a specified location, followed by three rotations about axes centered at the \((x, y, z)\) origin and parallel to the \((x_g, y_g, z_g)\) global coordinate directions
- 5 – the shell unit is positioned by positioning corner point 1 at a specified location, followed by three rotations about axes centered at corner point 1 and parallel to the \((x_g, y_g, z_g)\) global coordinate directions

**NROWS** reserved for future use; set equal to zero
**NCOLS**  reserved for future use; set equal to zero

**NLAYS**  element-layering parameter:

\[ \leq 0 \quad \text{generate one layer of standard (non-solid) elements} \]

\[ > 0 \quad \text{generate NLAYS layers of solid elements in the thickness direction (see N-1, on 6-34)} \]

**NFABS**  fabrication-specification parameter:

\[ \leq 1 \quad \text{one M-5 record is required} \]

\[ > 1 \quad \text{NFABS M-5 records required} \]

[go to M-2]
M-2 Shell Surface Constants

This record specifies properties of the reference surface. The type of surface is indicated by ISHELL (M-1). “Standard Shell Surfaces” on page 6-2 shows what these properties mean for the standard surface types defined by ISHELL = 2–12. M-2 is also read when a user-written LAME is used (ISHELL = 1). In that case, the properties have the following meaning:

- PROP(1) = X value for boundary line 1
- PROP(2) = X value for boundary line 3
- PROP(3) = Y value for boundary line 4
- PROP(4) = Y value for boundary line 2
- PROP(5) through PROP(8) may be used for other parameters in user-written LAME.

Boundary lines are numbered as shown in Figure 6.1 on page 6-19.

When ISHELL = 1 and IUGRID = 1 (N-1), the range of (X, Y) is automatically set to

\[
0 \leq X \leq 1 \quad \text{and} \quad 0 \leq Y \leq 1
\]

PROP remains intact, but it is not used by STAGS when IUGRID = 1 is set in conjunction with ISHELL = 1. In that case, the user may assign any values to the PROP array. See “LAME Reference Surface Geometry” on page 12-19.

| PROP(i), i=1,8 |

| PROP(i) | surface properties depending on ISHELL (M-1); all angles are in degrees for ISHELL = 2–12. |

Refer to “Standard Shell Surfaces” on page 6-2. For clarity of presentation, PROPs are shown as having positive values in the sketches there. Negative values are also permissible, except for radius values, such as \( R_a \), \( R_b \) in the Annular Plate. However, the following relations must be satisfied:

- PROP(2) > PROP(1)
- PROP(4) > PROP(3)
Some of these shell surfaces may include poles. **STAGS** will recognize the existence of a pole and will generate triangular elements at the pole under the conditions that are summarized in Table 6.1.

### Table 6.1 Triangular shell elements at poles.

<table>
<thead>
<tr>
<th>ISHELL</th>
<th>type</th>
<th>conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>annular plate</td>
<td>PROP(1) = 0</td>
</tr>
<tr>
<td>6</td>
<td>cone</td>
<td>PROP(5) or PROP(6) = 0</td>
</tr>
<tr>
<td>7</td>
<td>sphere</td>
<td>PROP(1) or PROP(2) = 0, PROP(1) or PROP(2) = 180, PROP(2) = 360</td>
</tr>
<tr>
<td>9</td>
<td>elliptic cone</td>
<td>PROP(5) or PROP(6) = 0</td>
</tr>
<tr>
<td>10</td>
<td>paraboloid</td>
<td>PROP(6) = 0</td>
</tr>
<tr>
<td>11</td>
<td>ellipsoid</td>
<td>PROP(1) or PROP(2) = 0, PROP(1) or PROP(2) = 180, PROP(2) = 360</td>
</tr>
</tbody>
</table>

**IGLOBE** (M-1) global position option

if \((\text{IGLOBE} = 2)\) then go to M-3
elseif \((\text{IGLOBE} = 3)\) then go to M-4a
elseif \((\text{IGLOBE} = 4 \text{ or } 5)\) then go to M-4d
else go to M-5
M-3 Shell Unit Orientation—Straight-Line Boundary

OBSOLETE

This option is retained for compatibility, but its use is not recommended due to the addition of the more general, more robust options IGLOBE = 3 – 5.

This record is used only if IGLOBE = 2 (M-1). It defines the \((x_g, y_g, z_g)\) global coordinates of one reference point on the surface of the shell unit. Any point can be chosen as long as it is not located on a line of juncture with a previously defined unit.

\[
\begin{array}{cccccc}
\text{NREP} & \text{NCEP} & \text{XGEP} & \text{YGEP} & \text{ZGEP} \\
\end{array}
\]

NREP row number of reference point
NCEP column number of reference point
XGEP, YGEP, ZGEP \((x_g, y_g, z_g)\) global coordinates of reference point

goto M-5
M-4a Shell Unit Orientation—Corner Point 1

The records M-4a, M-4b and M-4c are used only if IGLOBE = 3 (M-1). They define the global coordinates of the three corner points 1, 2, 3 on the shell unit. The numbering system is shown in Figure 6.1.

XGC1  YGC1  ZGC1

XGC1, YGC1, ZGC1 \( (x_g, y_g, z_g) \) global coordinates of corner point 1

Figure 6.1 Shell-unit boundary- and corner-numbering schemes.

go to M-4b
**M-4b Shell Unit Orientation—Corner Point 2**

\[ \text{XGC2, YGC2, ZGC2} \]

\[ (x_g, y_g, z_g) \] global coordinates of corner point 2

*go to M-4c*
M-4c Shell Unit Orientation—Corner Point 3

XGC3  YGC3  ZGC3

XGC3, YGC3, ZGC3  \((x_g, y_g, z_g)\) global coordinates of corner point 3

\(\Rightarrow\)  go to M-5
**M-4d Shell Unit Orientation—Translation**

Record M-4d is input if IGLOBE = 4 or 5 (M-1). These options permit the user to move a shell unit into position manually, even for the case where the corner points lie in a line, as is the case for closed shell units such as complete cylinders or cones. The shell unit is positioned by specifying the global coordinates of a reference point and then applying rotations about axes that are centered at the reference point and parallel to the global axes. Record M-4d provides the global coordinates for the reference point chosen. Record M-4e specifies the Euler angles for rotation.

If IGLOBE = 4, the reference point is the origin of the (x, y, z) branch coordinate system, shown in “Standard Shell Surfaces” on page 6-2 for each choice of ISHELL (M-1). This option has the effect of translating the shell unit in the global coordinate directions, with distances given by \((x_g, y_g, z_g) = (XG, YG, ZG)\), since \((x, y, z)\) is initially coincident with \((x_g, y_g, z_g)\), and the shell unit is fixed with respect to the \((x, y, z)\) system.

If IGLOBE = 5, the reference point is the node belonging to row 1, column 1 (corner point 1, Figure 6.1 on page 6-19). This option has the effect of translating the shell unit to position corner point 1 at the location \((x_g, y_g, z_g) = (XG, YG, ZG)\).

\[ XG \quad YG \quad ZG \]

\[ XG, YG, ZG \quad \text{a position vector in the } (x_g, y_g, z_g) \text{ global coordinate system, defining the location of the reference point} \]

\[ go \ to \ M-4e \]
M-4e Shell Unit Orientation—Rotation

This record consists of the Euler angles used to rotate the shell unit about the reference point (M-4d). If IGLOBE = 4, the reference point is the origin of the (x, y, z) branch coordinate system, and if IGLOBE = 5, the reference point is corner point 1. Note that the reference point was positioned in global space by the M-4d record.

The angles input here represent right-handed rotations about axes that are centered at the reference point and parallel to the global axes (x_g, y_g, z_g), in the following order:

- rotate first about the z_g axis, by an amount \( \phi \)
- rotate next about the y_g axis, by an amount \( \theta \)
- rotate last about the x_g axis, by an amount \( \chi \)

**CAUTION:** Note that the rotations are order-dependent and are applied in the reverse order of input.

Each rotation is counterclockwise with the viewer looking down along the chosen axis (i.e., in the negative coordinate direction) onto the plane of the remaining two axes. For the STAGS right-handed global coordinate system, the convention chosen is called the space-fixed system of right-hand rotations, given symbolically by the relation

\[
R(\chi, \theta, \phi) = R(\chi)R(\theta)R(\phi)
\]

where each symbol represents a 3 \times 3 orthogonal matrix.

---

**XGROT YGROT ZGROT**

**XGROT** the angle \( \chi \) (degrees)

**YGROT** the angle \( \theta \) (degrees)

**ZGROT** the angle \( \phi \) (degrees)

* go to M-5
**M-5 Shell Wall**

Typically, a shell unit defines a two-dimensional reference surface that is subsequently discretized into one- and/or two-dimensional finite elements. The properties of this surface are provided by the Wall Fabrication Table (see the K records, described above), by the GCP Fabrications Table (described in the I records, above), or by a user-written WALL subroutine. The M-5 record contains the necessary data linking a given fabrication to the reference surface. Data input here will also tell STAGS whether the strain-displacement relations are linear or nonlinear, whether there is to be nonlinear material behavior, and whether there are initial geometric imperfections defined either as a trigonometric expansion, in a user-written subroutine, or by scaled buckling modes (see records B-2 and B-5). A single M-5 record is always required for such shell units.

A shell unit that is constructed with type E830, E840 or E849 sandwich elements requires a separate fabrication identifier for each distinct component in the thickness direction of the shell unit. The simplest E830, E840 or E849 sandwich construction—which uses one E330, E410 or E480 thin-shell element for its lower face sheet, a single sandwich-core element for its core, and a second E330, E410 or E480 thin-shell element for its upper face sheet—requires three fabrication identifiers. The first of these is used for the lower face sheet, the second is used for the core, and the third is used for the upper face sheet. A more complex E830, E840 or E849 sandwich construction—with NLAYS > 1 core components at each station—requires NLAYS + 3 fabrication identifiers. The first of these is used for the lower face sheet, the second is used for the lower face sheet, the third is used for the next core element, ... and so on, and the next-to-last fabrication is used for the upper face sheet. STAGS uses the last (NFABSth) of these fabrications to construct a “fictitious” E330, E410 or E480 thin-shell element at each of the interfaces between successive layers of the core-section components. STAGS uses the reference surface of the shell unit to define the location of the interface between the lower face sheet and the first (lowest) sandwich-core component of a shell unit that uses E830, E840 or E849 sandwich elements. STAGS then constructs additional layers of nodes by adding the thickness of each successive sandwich-core component—the uppermost such layer defining the interface between the uppermost core element and the upper face sheet of the assembly.

A shell unit that is constructed with type E881, E882, E883 or E885 three-dimensional solid elements requires one fabrication identifier specifying the single fabrication to be used for all of those elements, or it requires a separate fabrication identifier for each layer of elements across the thickness of the shell: this is controlled by the NLAYS and NFABS parameters on the M-1
record, described above. With these types of “standard” solid elements, STAGS computes the total thickness of the shell wall by summing the thickness of each layer of the construction. STAGS uses half of this total thickness to determine the offset in the normal direction from the shell’s reference surface (which is specified via ISHELL on M-1 and PROPS on M-2) to the “lower” surface of the solid shell, and places the first layer of shell-unit nodes there. Successive additional layers of shell-unit nodes are placed on “parallel” nodal surfaces that are offset from this first surface by the thickness of each successive layer of elements—or by the appropriate whole-number fraction of that thickness, as appropriate. For these types of solid elements, then, the reference surface of the shell is always midway between its lower and upper surfaces.

The principal material axes for individual layers in a fabrication are defined on the K-2 record (ZETL) with the traditional STAGS definition of shell wall fabrication properties, and on the I-21d record (ANGSHEL) with the GCP definition. The user may also rotate the fabrication in the plane tangent to the reference surface by an angle ZETL (or ANGSHL) so that the angle a principal material axis makes with the shell coordinate lines is the sum of ZETL (or ANGSHL) and the angle defined for the individual layer in question. The default system for stress output is along the natural shell coordinates. Output can also be specified along an arbitrary direction defined by the user.

It is possible to suppress geometrically nonlinear terms for any particular unit using the ILIN parameter. For linear analysis or small vibrations (stress free), INDIC = 0 or 2 (B-1), ILIN is meaningless. For nonlinear analysis (including bifurcation or vibrations with nonlinear stress state and transient analysis) INDIC = 3, 4, 5 or 6, ILIN can be used to suppress nonlinear effects in some shell units and thus to expedite execution. For bifurcation buckling from a linear stress state (INDIC = 1) ILIN can be used to prevent buckling in certain shell units. ILIN must be 0 for at least one element or shell unit when performing a bifurcation analysis.

If nonlinear material behavior (plasticity or creep) is desired in a given shell unit, IPLAS must be greater than zero. Like ILIN, IPLAS allows the user to turn plasticity on or off without changing the contents of the material and wall construction libraries. When IPLAS=0, all material properties are constructed from the linear elastic data given on the I-2 records. Two options are given for plasticity. If IPLAS=1, the material law is satisfied at each element integration point, which for the vast majority of cases gives the most accuracy. If IPLAS=2, the material law is enforced only at the element centroid, with the same plastic strain distributed evenly to the integration points. The result is a lower-order, approximate solution requiring only a fraction of the plasticity computations needed when IPLAS=1. This option can be used when stresses are slowly varying over the structure. There is one important special case that should be mentioned. Some elements, like the E410, have a high-order bending interpolation and a lower-order membrane
interpolation. When there is eccentricity in a wall section (the variable ECZ on this record) that introduces coupling between the membrane and bending element stiffnesses, the resulting mismatch causes stress overshoot, with the stresses computed at the integration points potentially lying far above and below the average over the element. These oscillations are an artifact of the computation and do not represent real behavior. For elastic response, the displacements will not be adversely affected. For a plastic analysis, however, the path-dependent nature of the material response can cause large errors when significant plastic flow occurs. Since centroidal plasticity avoids these problems, we recommend setting IPLAS=2 when ECZ ≠ 0 and when using type E320, E330, E410, E420 and/or E430 elements (N-1).

Deterministic imperfections are defined in the form of trigonometric functions on data records if IWIMP>0, or in any form in a user-written DIMP subroutine if IWIMP = -1. Random imperfections are defined in terms of trigonometric functions if IRAMP=1. As they are generated independently for each shell unit, these shell-unit-specific imperfections may not exhibit continuity across shell unit boundaries. These imperfections are added to the buckling-mode imperfections that are generated when NIMPFS is nonzero (on the B-2 record).

As noted previously, there must be at least one M-5 record at this point in the user’s input file. If NFABS > 1 (M-1), then NFABS M-5 records are required.
IWALL  IWIMP  ZETA  ECZ  ILIN  IPLAS  IRAMP

### IWALL

Fabrication identifier:
- 0 – shell wall properties are given in user-written subroutine WALL
- >0 – shell wall configuration number in the Wall Fabrication Table (K-1)
- <0 – shell or solid fabrication identifier in the GCP Fabrications Table (on I-21) for shell elements or on I-22 for solid elements: *this option is required for E830, E840 and E849 sandwich-core elements*

When **IWALL** = 0, data below indicated by ✤ are defined in user-written subroutine WALL; they are automatically initialized to zero before WALL is called—superseding any nonzero values input here

### IWIMP

Imperfection option
- -1 – initial imperfections are defined in user-written DIMP
- 0 – no initial imperfections
- >0 – number of imperfection components defined on M-6 records; **IWIMP** ≤ 50

### ZETA

Angle ζ between the wall-reference coordinate x_w and the fabrication coordinate x; ζ is a right-handed rotation about z. See Figure 6.2 on page 6-28.

### ECZ

Eccentricity in Z′ direction. **ECZ** is the Z′ coordinate of the middle surface; see Figure 6.2 on page 6-28. Also, see “Effects of Eccentricity” on page 16-6.

### ILIN

Geometric nonlinearity flag:
- 0 – nonlinear strain-displacement relations
- 1 – linear strain-displacement relations

Note that with **ILIN** = 1, bifurcation buckling is suppressed in the shell unit

### IPLAS

Material nonlinearity flag:
- 0 – elastic behavior only
- 1 – plasticity included, with the material law satisfied at each element integration point
- 2 – plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)

### IRAMP

Random imperfections flag:
- 0 – no random imperfection used
- 1 – random imperfections used
NFABS (M-1) fabrication-specification parameter

if (fewer than NFABS fabrications have been specified) then
continue fabrication specification via additional M-5 record(s)
else
if (IWIMP > 0) then go to M-6
elseif (IRAMP = 1) then go to M-7a
else go to N-1
endif

Figure 6.2  Shell wall orientation—eccentricity (top) and reference angle (bottom). Variable wall thickness and eccentricity, shown here, require user-written subroutine WALL (see Chapter 12 “User-Written Subroutines”). Also, see Figure 5.6 on page 5-130.
M-6 Shell Imperfections

The M-6 records allow the user to define initial geometric imperfections in the shell reference surface. The imperfection consists of a deviation from the ideal geometry in the form of a displacement field \((u_0, v_0, w_0)\). The imperfect shell unit is considered stress-free under zero load. Each record represents one component of the imperfection and the total initial displacement is obtained through addition of all the components. The components are trigonometric functions in the two-dimensional \((X, Y)\) surface coordinate space, with maximum points located at \((X, Y) = (X_1, Y_1)\), and with half wavelengths equal to \(X_L\) in the \(X'\) direction and \(Y_L\) in the \(Y'\) direction. Notice that the dimensions of \(X_L, Y_L\) are the same as of \((X, Y)\), which depend upon the specific shell type (see “Standard Shell Surfaces” on page 6-2). For example, in a cylindrical shell, \(X\) is length, and \(Y\) is degrees.

The amplitude of each imperfection component, \(W_0\), can be expressed as

\[
W_0 = WAMP \cdot \cos \left( \frac{X - X_1}{X_L} \pi \right) \cdot \cos \left( \frac{Y - Y_1}{Y_L} \pi \right)
\]

The record is repeated \(IWIMP\) times (M-5).

| X1, Y1 | value of the \((X, Y)\) surface coordinates at one of the maximum points |
| XL    | half-wavelength of the imperfection in the \(X'\) direction; if \(XL = 0\), \(XL = \infty\) is set |
| YL    | half-wavelength of the imperfection in the \(Y'\) direction; if \(YL = 0\), \(YL = \infty\) is set |
| WAMP  | amplitude of imperfection component |
| ID    | component identifier: |
|       | 1 – \(u\) imperfection |
|       | 2 – \(v\) imperfection |
|       | 0,3 – \(w\) imperfection |

\[
\text{IRAMP (M-5)} \quad \text{random imperfections flag}
\]

if \(\text{IRAMP} = 1\) then go to M-7a
else go to N-1
**M-7a Random Imperfection Shapes**

This record is used only if IRAMP = 1 (M-5). It defines the shape of a number of components defining randomly-determined initial geometric imperfections. The user defines with the parameters N1 and N2 the number of half-wavelengths of a “basic imperfection mode” over the shell unit dimensions in the (X’, Y’) shell coordinate directions. The program includes imperfections in all modes that are of the form

\[ N = N_1 + i \cdot N_I \quad i = 1, 2, \ldots \quad N \leq N_2 \]
\[ M = M_1 + j \cdot M_I \quad j = 1, 2, \ldots \quad M \leq M_2 \]

where N2 and M2 represent the maximum number of halfwaves over the shell unit dimensions, and where N_I and M_I represent increments. The actual size of the imperfections can be printed. The components are harmonics in which both the magnitude and location of the maximum are independently chosen at random. Only transverse (wθ) random imperfections can be defined.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N_I</th>
<th>M1</th>
<th>M2</th>
<th>M_I</th>
<th>KTEST</th>
</tr>
</thead>
</table>

- N1: initial wave number in the X’ direction
- N2: final wave number in the X’ direction
- N_I: wave number increment in the X’ direction
- M1: initial wave number in the Y’ direction
- M2: final wave number in the Y’ direction
- M_I: wave number increment in the Y’ direction
- KTEST: print option:
  - 0 – no print of random imperfections
  - 1 – print random imperfections generated

*go to M-7b*
M-7b Random Imperfection Amplitudes

This record is included only if \textbf{IRAMP} > 0 (M-5). It gives the information on which the amplitudes of random imperfection components are based. The amplitude, $w_0$, of each component is set equal to

$$w_0 = R(N, M) \cdot \text{GRAMP} \cdot \frac{N_1 \cdot N_2}{N \cdot N} ^ \text{EXP}$$

where \text{GRAMP} and \text{EXP} are input variables and $R(N,M)$ is a random number with uniform probability in the interval $[-1, 1]$. The values of $R$ are chosen independently for the different components. The exponent \text{EXP} is introduced because the shorter waves in the imperfection pattern are likely to have smaller amplitudes, as in a Fourier expansion of a continuous periodic function. Presently little experience is available on a value for \text{EXP}. Based on continuity arguments alone, \text{EXP} should be at least unity for so-called “smooth” imperfections found in most applications. Although in most cases restart executions will not require passing through the model setup phase again, in some situations (such as load and boundary condition changes) the random imperfections will be redefined. To preserve the same random pattern for these restarts, set \text{RANK} to some nonzero number.

\textbf{GRAMP  RANK  EXP}

<table>
<thead>
<tr>
<th>GRAMP</th>
<th>general random amplitude factor; random numbers will be chosen in the range $[-\text{GRAMP, GRAMP}]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>RANK</td>
<td>random number initialization constant (seed)</td>
</tr>
<tr>
<td>EXP</td>
<td>exponent used to scale the amplitude of imperfection components corresponding to higher harmonics</td>
</tr>
</tbody>
</table>

\textit{go to N-1}
6.2 Discretization

The process of discretizing a shell unit into a number of finite elements is performed in two stages: (1) Grid Generation and (2) Mesh Generation.

Grid Generation

The user defines an “underlying grid” composed of “rows” and “columns,” the intersection of which are called “grid points.” The grid points (or “nodes”) are generated automatically such that rows and columns correspond to lines of constant $X$ and $Y$, respectively. The computational freedoms ($dof$) at a grid point are in computational coordinates $(x'', y'', z'')$, which are identical to shell coordinates $(X', Y', Z')$ in a shell unit.

The spacing of rows and columns for either of the above options may be

- uniform
- segmented, i.e., piecewise uniform
- variable

where spacing refers to $\Delta X$ or $\Delta Y$ for the standard mesh and to $\Delta S$ or $\Delta T$ for the specialized mesh. The two mesh types are described next.

Mesh Generation

The user introduces an “overlying mesh” of finite elements, utilizing the underlying grid in one of the following ways:

Standard Mesh

Shell or plate elements are automatically introduced utilizing all underlying grid points. The user is required to select a basic “triangular” or “quadrilateral” element type from the element library (Chapter 14 “The Element Library”) to be used throughout the mesh. When quad elements are selected and the shell unit has a pole, STAGS substitutes triangles at the pole as and if required. Additionally, a “cutout” may be specified as a range of rows and columns within which no elements will be introduced.
**Specialized Mesh — INACTIVE**

Shell or plate elements are introduced in user-controlled “patches.” Each patch requires the definition of an element type, grid point connections of the first element, and some repetition parameters. By incorporation of the transition elements (X-ref) it is therefore possible to utilize a uniformly-spaced underlying grid to generate a mesh of variable refinement. Furthermore, it is acceptable to use any of the grid generating options as a basis for the overlying mesh, i.e., for element generation. Grid points which are not utilized by elements remain inactive during the analysis and do not contribute any computational overhead.

Parts of the grid can be left open, i.e., no elements are defined. In that case, these parts will represent cutouts.

Regardless of discretization procedure, the user should be aware of the following restrictions:

- The underlying grids of connected units must match, i.e., have an equal number of grid points, along the juncture line.
- Adjacent elements should be of “consistent” types to achieve best results, i.e., convergence and accuracy of the solution. Consistency, here, means that the elements have identical nodal and midside freedoms.
N-1 Discretization Control

This record establishes the “grid” and “mesh” generating options to be used in the discretization process. Reference will be made to the preceding paragraphs, which define these options, and to Chapter 14 “The Element Library”.

When a standard mesh is employed, the user’s choice for the KELT parameter on the N-1 record instructs STAGS (in most cases) to generate the element mesh for the shell unit utilizing as many shell, sandwich or solid elements of type KELT as may be required. In some cases, the (optional) KELTX parameter may also be used to specify an alternate triangular element type or common-side direction information, as described below. In any event, the KELT parameter must be specified and must be one of the following values: 320, 330, 410, 411, 420, 430, 480, 830, 840, 849, 881, 882, 883 or 885.

The choice of KELT=320 (or KELT=330) instructs STAGS to use two adjacent E320 triangular elements (or two adjacent E330 triangular elements) in each quadrilateral cell and to use a single E320 element (or a single E330 element) in each triangular cell of the shell unit being created. The KELTX parameter may be used here to specify the orientations of the common sides of these elements, as described below. If KELTX is omitted or set equal to zero, STAGS alternates the common-side direction as shown in sketch (a) in the following figure:

![Figure 6.3](image)

(a) $\text{KELTX} = 0$  (b) $\text{KELTX} = 1$  (c) $\text{KELTX} = 2$

**Figure 6.3** Influence of KELTX on E320 or E330 common-side directions.
The choice of $\texttt{KELT}=410$ (or $\texttt{KELT}=411$) instructs STAGS to use a single $\texttt{E410}$ (or $\texttt{E411}$) quadrilateral element in each quadrilateral cell of the shell unit and to use a single $\texttt{E320}$ or $\texttt{E330}$ triangular element in each pole-imposed triangular cell. The $\texttt{KELTX}$ parameter may be utilized here to specify the user’s choice of $\texttt{E320}$ or $\texttt{E330}$ for the triangular element(s) to be used here. When transition elements are generated for an $\texttt{E410}$-type shell unit (as described below), the $\texttt{KELT}$ parameter specifies the basic element type ($\texttt{E410}$) for the shell unit; and connectivity considerations on the boundary (or boundaries) dictate the type(s) of transition elements that are to be used.

The choice of $\texttt{KELT}=420$ (or $\texttt{KELT}=430$) instructs STAGS to use two adjacent $\texttt{E320}$ triangular elements (or two adjacent $\texttt{E330}$ triangular elements) in each quadrilateral cell and to use a single $\texttt{E320}$ (or $\texttt{E330}$) element in each triangular cell of the shell unit being created. Setting $\texttt{KELT}=420$ (or $\texttt{KELT}=430$) gives exactly the same results as the $\texttt{KELT}=320$ (or $\texttt{KELT}=330$) choices described above.

The choice of $\texttt{KELT}=480$ instructs STAGS to use a single $\texttt{E480}$ quadrilateral element in each 9-node quadrilateral cell of the shell unit (the nodal mesh for which must therefore have an odd number of rows and an odd number of columns). The current version of STAGS does not accommodate poles in or transitions involving $\texttt{E480}$-type shell units, so the $\texttt{KELTX}$ and the $\texttt{MESH1}$, $\texttt{MESH2}$, $\texttt{MESH3}$ and $\texttt{MESH4}$ parameters should not be used.

The choice of $\texttt{KELT}=830$ instructs STAGS to use a single- or multi-layered $\texttt{E830}$ sandwich construction (of alternating 3-node triangular thin-shell and 6-node triangular sandwich-core elements) for the shell unit in question. STAGS does not accommodate mesh transitions with $\texttt{E830}$-type shell units.

The choice of $\texttt{KELT}=840$ instructs STAGS to use a single- or multi-layered $\texttt{E840}$ sandwich construction (of alternating 4-node quadrilateral thin-shell and 8-node sandwich-core elements) for the shell unit in question. The current version of STAGS does not accommodate poles in $\texttt{E840}$-type shell units, so the $\texttt{KELTX}$ parameter should not be used. STAGS does accommodate mesh transitions with $\texttt{E840}$-type shell units, however; so STAGS does use the $\texttt{MESH1}$, $\texttt{MESH2}$, $\texttt{MESH3}$ and $\texttt{MESH4}$ parameters to generate type $\texttt{E845}$ and/or type $\texttt{E847}$ sandwich transition elements along the edge(s) of the shell unit, as and if required.

The choice of $\texttt{KELT}=849$ instructs STAGS to use a single- or multi-layered $\texttt{E849}$ sandwich construction (of alternating 9-node thin-shell and 18-node sandwich-core elements) for the shell unit in question. STAGS does not accommodate mesh transitions with $\texttt{E849}$-type shell units.

The choice of $\texttt{KELT}=881$ instructs STAGS to use one or more $\texttt{E881}$ solid elements through the thickness of each multi-layered quadrilateral cell of the shell unit. The current version of STAGS does not accommodate poles or transitions with $\texttt{E881}$-type shell units, so the $\texttt{KELTX}$ $\texttt{MESH1}$, $\texttt{MESH2}$, $\texttt{MESH3}$ and $\texttt{MESH4}$ parameters should not be used.
The choice of \texttt{KELT}=882 (or \texttt{KELT}=883 or \texttt{KELT}=885) instructs \textsc{stags} to use one or more \texttt{E882} (or \texttt{E883} or \texttt{E885}) solid elements through the thickness of each multi-layered 9-node quadrilateral cell of the shell unit. The current version of \textsc{stags} does not accommodate poles or transitions with these types of shell units, so the \texttt{KELTX MESH1}, \texttt{MESH2}, \texttt{MESH3} and \texttt{MESH4} parameters should not be used.

With the \texttt{E410} element the user has the \texttt{INTEG} = 1 option to use a modified form of Gaussian integration in which the element centroid is added as the fifth integration point. The added point may, in some situations, suppress spurious strain-free deformation modes (mechanisms) that can crop up in cases involving intersecting free boundaries. In most cases, careful choice of boundary conditions will avoid such difficulties, making unnecessary the extra integration point that increases computational expense and decreases the rate of convergence as the grid is refined. For the \texttt{E411} elements, a full $3 \times 3$ Gaussian integration is provided to prevent spurious mechanisms. If the boundary conditions are known to prevent strain-free deformation modes, the user should choose the standard reduced $2 \times 2$ integration.

In the \texttt{E410} and \texttt{E411} elements, the bending strain energy is based on a Taylor series with 12 degrees of freedom (corresponding to the two in-plane rotations and the out-of-plane translation at each corner). The series must include two fourth order terms, $x^3y$ and $xy^3$, which can lead to a small discontinuity between elements along a given edge. The user can essentially eliminate this discontinuity in the lateral displacements by requesting the use of a penalty function ($\texttt{IPENL} = 1$), which will make the element stiffer and could be helpful if the strain energy converges from below as the grid is refined.

When, for an \texttt{E410}-type shell unit, any of the variables \texttt{MESH1-MESH4} is set equal to 1 \textsc{stags} will flag every other node as dependent along the corresponding boundary. \textsc{stags} will then automatically generate \texttt{E510} or \texttt{E710} transition elements along that boundary to constrain out the dependent nodes. The \texttt{E510} element is used where a single dependent node occurs in an \texttt{E410} element. The \texttt{E710} element is used in a corner where refinement is requested along two adjacent lines (for example, if \texttt{MESH2} and \texttt{MESH3} are both set to 1). For more information about this, see “\texttt{E510 and E710 Quadrilateral mesh-transition shell elements}” on page 14-25.

Similarly, when any of the variables \texttt{MESH1-MESH4} is set equal to 1 for an \texttt{E840}-type shell unit, \textsc{stags} will flag every other node as dependent along the corresponding boundary and will then automatically generate \texttt{E845} or \texttt{E847} transition sandwich elements along that boundary to constrain out the dependent nodes. The \texttt{E845} element is used where a single dependent node occurs in an \texttt{E840} element. The \texttt{E847} element is used in a corner where refinement is requested along two adjacent lines (for example, if \texttt{MESH2} and \texttt{MESH3} are both set to 1). For more information about this, see “\texttt{E845 and E847 mesh-transition sandwich elements}” on page 14-46.
Discretization Model Input—Shell Units

Transport Model Input—Shell Units

KELT NNX NNY IRREG IUGRID INTEG IPENL MESH1 MESH2 MESH3 MESH4 KELTX

KELT  element code number (see Chapter 14 “The Element Library”)

>0  –  standard mesh:

  =  320  E320  3-node C\(^1\) triangular shell element
  =  330  E330  3-node C\(^0\) MIN3 triangular shell element
  =  410  E410  4-node C\(^1\) quadrilateral shell element
  =  411  E411  4-node C\(^1\) quadrilateral shell element
  =  420  pairs of E320  3-node triangular shell elements
  =  430  pairs of E330  3-node triangular shell elements
  =  480  E480  9-node C\(^0\) ANS quadrilateral shell element
  =  830  E830  6-node wedge-shaped sandwich solid/shell element
  =  840  E840  8-node hexahedral-shaped quadrilateral sandwich solid/shell element
  =  849  E849  18-node hexahedral-shaped quadrilateral sandwich solid/shell element
  =  881  E881  8-node ANS hexahedral solid element
  =  882  E882  18-node ANS hexahedral solid element
  =  883  E883  27-node ANS hexahedral solid element
  =  885  E885  20-node displacement-based hexahedral solid element

0  –  specialized mesh — INACTIVE

NNX  indicates row spacing of underlying grid; refers to the X coordinate

  0  –  uniform spacing in X direction
  -1  –  variable spacing, X coordinates defined on N-4 record
  >0  –  NNX defines the number of segments in the X direction,
       each with constant spacing; NNX \leq 30

NNY  indicates column spacing of underlying grid; refers to the Y coordinate

  0  –  uniform spacing in Y direction
  -1  –  variable spacing, Y coordinates defined on N-7 record
  >0  –  NNY defines the number of segments in the Y direction,
       each with constant spacing; NNY \leq 30

IRREG  mesh irregularities

  0  –  standard mesh with no irregularities
  >0  –  standard mesh with IRREG cutouts (to be defined on N-8 records)
       (not operational with sandwich or solid elements)
**IUGRID**

$(X, Y)$ surface coordinate range flag; relevant only when $\text{ISHELL} = 1 \ (M-1)$.

By default, when $\text{ISHELL} = 1$, $\text{PROP} \ (M-2)$ is used to set the range of $(X, Y)$.

$$\text{PROP}(1) \leq X \leq \text{PROP}(2) \quad \text{PROP}(3) \leq Y \leq \text{PROP}(4)$$

However, when $\text{IUGRID} = 1$, the range of $(X, Y)$ is set to

$$0 \leq X \leq 1 \quad 0 \leq Y \leq 1$$

$\text{PROP}$ remains intact, but is not used by $\text{STAGS}$ when $\text{IUGRID} = 1$. In that case, the user may assign any values to the $\text{PROP}$ array, which is available in user-written subroutine $\text{LAME}$, used when $\text{ISHELL} = 1$.

Set $\text{IUGRID} = 0$ in all other cases.

**INTEG**

integration type:

- $0$ – standard integration:
  - $2 \times 2$ Gauss points, for $\text{E410}$, $\text{E411}$ elements and $\text{E840}$ face sheets
  - $2 \times 2 \times 2$ Gauss points (reduced integration), for $\text{E885}$ elements
- $1$ – modified 5-point integration, for $\text{E410}$ and $\text{E411}$ elements
- $2$ – full $3 \times 3$ Gauss integration for $\text{E411}$ element;
  - $3 \times 3 \times 3$ Gauss points, for $\text{E885}$ elements (not recommended)

**IPENL**

penalty option:

- $0$ – no penalty function on fourth-order terms in elements $\text{E410}$ and $\text{E411}$
- $1$ – penalty function included in elements $\text{E410}$ and $\text{E411}$

**MESH1**

governs generation of dependent nodes along boundary line 1

- $0$ – no dependent nodes generated (the mesh is continuous between shell units along boundary line 1).
- $1$ – generate dependent nodes along boundary line 1 (the mesh doubles between shell units along boundary line 1), this facilitates the use of $\text{E510}$ and/or $\text{E710}$ quadrilateral shell transition elements on boundary 1.
- $3$ – generate dependent nodes along boundary line 1 (the mesh doubles between shell units along boundary line 1), this facilitates the use of $\text{E330}$ triangular shell transition elements on boundary 1.
MESH2
governs generation of dependent nodes along boundary line 2

0  –  no dependent nodes generated (the mesh is continuous between shell units along boundary line 2).

1  –  generate dependent nodes along boundary line 2 (the mesh doubles between shell units along boundary line 2), for E510 and/or E710 quadrilateral shell transition elements.

3  –  generate dependent nodes along boundary line 2 (the mesh doubles between shell units along boundary line 2), for E330 triangular shell transition elements.

MESH3
governs generation of dependent nodes along boundary line 3

0  –  no dependent nodes generated (the mesh is continuous between shell units along boundary line 3).

1  –  generate dependent nodes along boundary line 3 (the mesh doubles between shell units along boundary line 3), for E510 and/or E710 quadrilateral shell transition elements.

3  –  generate dependent nodes along boundary line 3 (the mesh doubles between shell units along boundary line 3), for E330 triangular shell transition elements.

MESH4
governs generation of dependent nodes along boundary line 4

0  –  no dependent nodes generated (the mesh is continuous between shell units along boundary line 4).

1  –  generate dependent nodes along boundary line 4 (the mesh doubles between shell units along boundary line 4), for E510 and/or E710 quadrilateral shell transition elements.

3  –  generate dependent nodes along boundary line 4 (the mesh doubles between shell units along boundary line 4), for E330 triangular shell transition elements.

KELTX
auxiliary element code or adjacent-element direction index:

if KELT = 420 or 430, then
  KELTX = 320 — use type E320 element(s) at pole
  KELTX = 330 — use type E330 element(s) at pole

if KELT = 320 or 330, then
  KELTX = 0 — alternate common-side directions throughout the mesh
  KELTX = 1 — constant common-side direction, not intersecting (0,0)
  KELTX = 2 — constant common-side direction, intersecting (0,0)
if (NNX > 0) then go to N-2
endif (NNX = -1) then go to N-4
elseif (NNY > 0) then go to N-5
elseif (NNY = -1) then go to N-7
elseif (IRREG = 1) then go to N-8
else follow instructions at end of N-8
**N-2** X–Segment Length

This record is included only if \( \text{NNX} > 0 \) (N-1).

\[
\text{SEGLX}(i), \ i=1,\text{NNX}
\]

\( \text{SEGLX}(i) \) length of segment \( i \)

**Note:** Segment lengths are in the user’s units except when \( \text{ISHELL} = 3 \) (quadrilateral plate); for that case, the coordinates are scaled to the range \([0,1]\), so that \( \text{SEGLX}(i) \) are fractions of the unit interval.

\( \ast \)  \text{go to N-3} \
**N-3 X-Segment Spacing**

This record is included only if $\text{NNX} > 0$ (N-1).

<table>
<thead>
<tr>
<th>NSEGX(i), $i=1,\text{NNX}$</th>
</tr>
</thead>
</table>

$\text{NSEGX(i)}$  
number of intervals within segment $i$

*follow instructions at end of N-4*
**N-4 X–Coordinate**

This record is used to read in a sequence of X coordinates. It is included only if $\text{NNX} = -1$ (N-1).

---

**X(i), i = 1, NROWS**

- **X(i)**  
  X value for row $i$; must be monotonically increasing.
  
- **NROWS**, number of rows, was read on F-1 record.

**Note:** Coordinate values are in the user’s units except when $\text{ISHEL} = 3$ (quadrilateral plate); for that case, the coordinates are scaled to the range $[0,1]$, so that the $X(i)$ are points on the unit interval.

```
if (NNY > 0) then go to N-5
elseif (NNY = -1) then go to N-7
elseif (IRREG = 1) then go to N-8
else follow instructions at end of N-8
```
N-5 Y–Segment Length

This record is included only if $N_{NY} > 0$ (N-1).

$SEGLY(j), j = 1, N_{NY}$

$SEGLY(j)$  length of segment $j$

Note: Segment lengths are in the user’s units except when $ISHELL = 3$ (quadrilateral plate); for that case, the coordinates are scaled to the range $[0,1]$, so that $SEGLY(i)$ are fractions of the unit interval.

$\rightarrow$ go to N-6
N-6 Y-Segment Spacing

This record is included only if NNY > 0 (N-1).

NSEGY(j), j = 1, NNY

NSEGY(j) number of intervals within segment j

follow instructions at end of N-7
**N-7 Y–Coordinate**

This record is used to read in a sequence of $Y$ coordinates. It is included only if $N_{NY} = -1$ (N-1).

**Y(j), $j = 1$, NCOLS**

$Y(j)$: $Y$ value for column $j$; must be monotonically increasing.

$NCOLS$, number of columns, was read on an F-1 record.

**Note:** Coordinate values are in the user’s units except when $ISHELL = 3$ (quadrilateral plate); for that case, the coordinates are scaled to the range $[0,1]$, so that the $Y(i)$ are points on the unit interval.

```
if (IRREG = 1) then  go to  N-8
else  follow instructions at end of  N-8
```
N-8 Mesh Irregularity

This record is included only if IRREG > 0 (N-1). It is used to define the bounding grid lines of cutouts. Figure 6.4 shows a cutout in the shell surface located at boundary line 4 and rectangular in the X, Y space (i.e., the edge follows two intersecting grid lines). Here, NRW1, NRW2, NCL1, and NCL2 are row and column numbers corresponding to cutout boundaries. Cutouts of more general shape can be introduced by setting the shell thickness equal to zero inside the cutout with user-written subroutine WALL,

Figure 6.4  Shell unit with cutout.

A total of IRREG cutouts may be defined, and the cutouts may overlap, so that the resulting combined cutout is the union of all cutouts. This record is repeated IRREG times.
NRW1  NRW2  NCL1  NCL2

NRW1  row number of one edge of cutout
NRW2  row number of opposite edge of cutout; NRW2 > NRW1
NCL1  column number of one edge of cutout
NCL2  column number of opposite edge of cutout; NCL2 > NCL1

NSTFS (B-2)  number of shell units with discrete stiffeners
IUNIT (F-2)  unit currently being defined
NRGS (F-2)  number of rings in IUNIT
NSTR (F-2)  number of stringers in IUNIT

if (NRGS > 0) then go to O-1a
elseif (NSTR > 0) then go to O-2a
else go to P-1
6.3 Discrete Stiffeners

Discrete stiffeners may be

- parallel to lines with constant $X$; such stiffeners are sometimes referred to as rings
- parallel to lines with constant $Y$; such stiffeners are sometimes referred to as stringers

Discrete stiffener records (O-1a through O-3b) define the location, eccentricity, and property identification of each stiffener. Cross section properties are specified by reference to the data tables defined on the $J$ records.

Within each shell unit discrete stiffeners are assigned a number given by the order in which they are defined.

If the cross-section number is set to -1, the “moving plane boundary condition” is invoked. Discussed in detail under “B-2 General Model Summary” on page 5-12, this multi-point constraint is like a symmetry boundary, except that the plane of symmetry is allowed to rotate and translate as a rigid body. Each moving plane boundary is treated exactly like a ring or stringer, and counted as such (B-2 and F-2). Moving plane boundaries can be superimposed on real stiffeners, and real stiffeners can be stacked (please note the comments about plasticity and cross-section fabrication under “J-1 Cross-Section” on page 5-118).
O-1a Discrete Ring—Record 1

A ring is a stiffener which runs along a line of constant $X'$. This record assigns a cross-section to a ring stiffener by reference to the Data Tables (see Section 5.6 “Data Tables” on page 5-57). Eccentricity of beam axis and orientation of beam cross-section are also specified. Location is specified on O-1b. The O-1a/b sequence is repeated NRGS times (F-2).

\[ \text{ICROSS XSI ECY ECZ ILIN IPLAS} \]

**ICROSS**

- cross-section number, as defined by ITAB (J-1) in the Cross Section Table:
  - ICROSS = 0 indicates that the stiffener is defined in user-written CROSS
  - ICROSS = -1 indicates that the stiffener line is a moving plane boundary

When ICROSS = 0, data below indicated by ✤ are defined in CROSS. They are automatically initialized to zero before CROSS is called, thereby superseding any nonzero values input here. When ICROSS = -1, only ECY is relevant. Refer to Figure 6.5 on page 6-52 for an illustration of XSI, ECY, and ECZ.

- **XSI** angle $\xi$, in degrees, between the shell normal $Z'$ and the cross section $z$.
  $\xi$ is a right-handed rotation about $\tau$, the longitudinal axis of the ring, which is parallel to $Y'$.

- **ECY** eccentricity in the $X'$ direction. ECY is the $X'$ coordinate of the ($X', Z'$) pair which positions the origin of the ($Y, Z$) system. For ICROSS = -1 (moving plane boundary), ECY is a scale factor used for the numerical conditioning of Lagrange constraints introduced by the multi-point constraint. In that case, ECY should be of the same order of magnitude as entries in the stiffness matrix. A good guess is the modulus of elasticity of the material.

- **ECZ** eccentricity in the $Z'$ direction. ECZ is the $Z'$ coordinate of the ($X', Z'$) pair which positions the origin of the ($Y, Z$) system.

- **ILIN** geometric nonlinearity flag:
  - 0 – nonlinear strain-displacement relations
  - 1 – linear strain-displacement relations
IPLAS

material nonlinearity flag (see M-5):

0  –  linear elastic constitutive relations
1  –  plasticity included
2  –  centroidal plasticity

go to O-1b
Figure 6.5  Eccentricity and orientation of stiffener cross sections.

Compare with Figure 5.5 “Beam cross sections.” on page 5-122.
Also, see Section 16.2 “Beam Results” on page 16-8.
O-1b Discrete Ring—Record 2

This record specifies the location on the shell surface of a discrete ring. The stiffener location can be defined either by its row number and the column numbers at its end points or by corresponding surface coordinate values \((X, Y)\). Use of surface coordinates is convenient if the grid is changed in subsequent runs. If the coordinates \((XR, YR1, YR2)\) do not coincide with a grid line, \textsc{Stags} will select the closest grid line. \((XR, YR1, YR2)\) are ignored unless corresponding row or column numbers \((IR, JR1, JR2)\) are set to zero.

\begin{verbatim}
IR  JR1  JR2  XR  YR1  YR2
\end{verbatim}

- **IR**: row number of ring; \(IR = 0\) indicates that the location is given by the coordinate value \(XR\)
- **JR1**: column number at beginning of ring; \(JR1 = 0\) indicates that this location is given by the coordinate value \(YR1\)
- **JR2**: column number at end of ring \((JR2 > JR1)\); \(JR2 = 0\) indicates that this location is given by the coordinate value \(YR2\)
- **XR**: \(X\) coordinate of ring; ignored if \(IR > 0\)
- **YR1**: \(Y\) at beginning of ring; ignored if \(JR1 > 0\)
- **YR2**: \(Y\) at end of ring \((YR2 > YR1)\); ignored if \(JR2 > 0\)

\[\text{NSTFS} \quad (\text{B-2}) \quad \text{number of shell units with discrete stiffeners}\]
\[\text{IUNIT} \quad (\text{F-2}) \quad \text{unit currently being defined}\]
\[\text{NRGS} \quad (\text{F-2}) \quad \text{number of rings in IUNIT}\]
\[\text{NSTR} \quad (\text{F-2}) \quad \text{number of stringers in IUNIT}\]

\[
\text{if} \quad \left( \text{NRGS rings have been defined} \right) \quad \text{then}
\text{if} \quad \left( \text{NSTR > 0} \right) \quad \text{then go to O-2a}
\text{else} \quad \text{go to P-1}
\text{else} \quad \text{continue defining O-1a/b}
\]

STAGS 5.0 User Manual April, 2009 6-53
**O-2a Discrete Stringer—Record 1**

A stringer is a stiffener which runs along a line of constant $Y'$. This record assigns a cross-section to a stringer by referencing the Data Tables (see Section 5.6 “Data Tables”). Eccentricity of beam axis and orientation of beam cross-section are also specified. Location is specified on O-2b. The O-2a/b sequence is repeated $\text{NSTR}$ times (F-2).

---

**ICROSS XSI ECY ECZ ILIN IPLAS**

**ICROSS**
cross-section number, as defined by $\text{ITAB}$ (J-1) in the Cross Section Table:

$\text{ICROSS} = 0$ indicates that the stiffener is defined in user-written CROSS.

$\text{ICROSS} = -1$ indicates that the stiffener line is a moving plane boundary.

When $\text{ICROSS} = 0$, data below indicated by $\textcolor{red}{	ext{✤}}$ are defined in CROSS. They are automatically initialized to zero before CROSS is called, thereby superseding any nonzero values input here. When $\text{ICROSS} = -1$, only $\text{ECY}$ is relevant. Refer to Figure 6.5 on page 6-52 for an illustration of $\text{XSI}$, $\text{ECY}$, and $\text{ECZ}$.

$\textcolor{red}{	ext{✤}} \ \text{XSI}$
angle $\xi$, in degrees, between the shell normal $Z'$ and the cross section $\zeta$. $\xi$ is a right-handed rotation about $\hat{x}$, the longitudinal axis of the stringer, which is parallel to $X'$.

$\textcolor{red}{	ext{✤}} \ \text{ECY}$
eccentricity in the $Y'$ direction. $\text{ECY}$ is the $Y'$ coordinate of the $(Y', Z')$ pair which positions the origin of the $(\bar{y}, \bar{z})$ system. For $\text{ICROSS} = -1$ (moving plane boundary), $\text{ECY}$ is a scale factor used for the numerical conditioning of Lagrange constraints introduced by the multi-point constraint. In that case, $\text{ECY}$ should be of the same order of magnitude as entries in the stiffness matrix. A good guess is the modulus of elasticity of the material.

$\textcolor{red}{	ext{✤}} \ \text{ECZ}$
eccentricity in the $Z'$ direction. $\text{ECZ}$ is the $Z'$ coordinate of the $(Y', Z')$ pair which positions the origin of the $(\bar{y}, \bar{z})$ system.

$\textcolor{red}{	ext{✤}} \ \text{ILIN}$
geometric nonlinearity flag:

0 – nonlinear strain-displacement relations

1 – linear strain-displacement relations
IPLAS material nonlinearity flag (see M-5):

- 0  – linear elastic constitutive relations
- 1  – plasticity included
- 2  – centroidal plasticity

go to O-2b
O-2b Discrete Stringer—Record 2

This record specifies the location on the shell surface of a discrete stringer. The stiffener location can be defined either by its column number and the row numbers at its end points or by corresponding surface coordinate values \((X, Y)\). Use of surface coordinates is convenient if the grid is changed in subsequent runs. If the coordinates \((YS, XS1, XS2)\) do not coincide with a grid line, STAGS will select the closest grid line. \((YS, XS1, XS2)\) are ignored unless corresponding row or column numbers \((JS, IS1, IS2)\) are set to zero.

\[
\begin{array}{cccccc}
JS & IS1 & IS2 & YS & XS1 & XS2 \\
\end{array}
\]

- **JS**: column number of stringer; \(JS = 0\) indicates that the location is given by the coordinate value \(YS\)
- **IS1**: row number at beginning of stringer; \(IS1 = 0\) indicates that this location is given by the coordinate value \(XS1\)
- **IS2**: row number at end of stringer; \((IS2 > IS1); IS2 = 0\) indicates that this location is given by the coordinate value \(XS2\)
- **YS**: \(Y\) coordinate of stringer; ignored if \(JS > 0\)
- **XS1**: \(X\) at beginning of stringer; ignored if \(IS1 > 0\)
- **XS2**: \(X\) at end of stringer; \((XS2 > XS1); \) ignored if \(IS2 > 0\)

- **NSTFS** (B-2): number of shell units with discrete stiffeners
- **IUNIT** (F-2): unit currently being defined
- **NSTR** (F-2): number of stringers in **IUNIT**

if \(\text{NSTR strings have been defined}\) then \(\text{go to P-1}\)
else \(\text{continue defining O-2a/b}\)
6.4 Boundary Conditions

Records P-1 through P-4 are used to define constraints along boundary lines. Certain frequently-occurring boundary conditions may be defined on these records. For other types of constraints such as those applied at interior points, it is sometimes possible to use the load records (Q-1, Q-2, Q-3), defined on pages 6-67, 6-69 and 6-70. For cases with more complicated constraints, a user-written subroutine UCONST is included, or the constraints can be defined on data records as indicated on record B-2 (see, for example, the G-2 and G-3 records).

The unknowns at a juncture line are represented by the displacement components in the direction of the shell coordinates \((X', Y', Z')\) on the shell unit with the lowest number of those involved in the juncture. In general, the constraints are most readily introduced on the shell unit with the lowest number. Some care must be exercised when boundary conditions (P-1 through P-4 records) are introduced on the boundary lines that meet the juncture line to make sure that this does not introduce undue constraints. A case in point involves an annulus with a pole (radius of hole is zero): boundary line 1 (that has zero length) must be left unconstrained, \(\text{IBLN}(1) = 3\), since the single node at the pole already satisfies the boundary conditions imposed along the two intersecting edges 2 and 4.

Boundary lines are numbered 1–4 as shown in Figure 6.6 on page 6-58. This figure also shows the positive directions for displacement components. \((u, v)\) are in the tangent plane of the shell, \(w\) is in the direction of the shell normal, and \((ru, rv, rw)\) are the corresponding rotations. \((u, v, w)\) are in the \((X', Y', Z')\) shell coordinate directions. This holds at every node (both interior and boundary nodes) for all shell types, including the user-generated shell unit (\(\text{ISHELL} = 1, \text{M-1}\)).

Note that the condition of antisymmetry is identical to the simple support condition. This boundary condition as it stands causes problems for nonlinear executions involving very large rotations; at the time of publication, this boundary condition is being generalized to handle arbitrary response conditions. See Table 6.2 for a summary of boundary condition types.

Do not define constraints that are in conflict with displacements defined on load records. For example, the simple support boundary condition cannot be used together with a specified tangential displacement. A load applied on a clamped edge is ignored. Do not allow rigid body motion in a static analysis. If the boundary conditions as defined here allow rigid body motions, it is usually best to constrain the structure at suitably selected points by use of the load records (Q-1, Q-2, Q-3).
It is possible to define different boundary conditions for the pre-critical stress state (basic BC) and the incremental state (incremental BC) in the case of the bifurcation buckling analysis by setting INCBC = 1 (B-1). Incremental BC are meaningful for bifurcation buckling analyses (INDIC = 1 or 4, B-1) only.

When incremental BC are included (INCBC = 1), the default condition is that they are the same as the basic BC. Default incremental BC may be selectively overridden in two ways—by setting IBOND = 1 (P-1), and by defining load system B with specified displacements (LT = -1, Q-3).
In this situation, the value of the displacement \( P \), on the Q-3 record) is ignored, and a dof constraint (\textit{i.e.}, specified zero) is applied to the incremental BC. When different boundary conditions for buckling have been chosen, specified displacements in load system A (Q-1–Q-3) apply to the pre-critical stress state but not to the buckling solution, while specified-zero displacements in load system B apply to the buckling solution only. In such cases, a starting load factor for load system B (\texttt{STLD(2)}, C-1) is not needed. This is of particular concern with finite element models using solid and/or sandwich elements.

Lagrangian constraints (G-3/4 or UCONST) and partial compatibility (G-2) apply to both pre-critical and buckling analyses.

In an eigenvalue analysis with a nonlinear stress state, incremental BC are not necessary because it is much more straightforward to perform the nonlinear stress analysis in a separate run and to modify the P-1 and P-2 records in the subsequent eigenvalue analysis. Modifications to boundary conditions and loading are always permitted before restart.

### Table 6.2 Boundary condition (BC) summary; refer to Figure 6.6.

See P-2 record for an explanation of \texttt{ITRA}, \texttt{IROT}.

<table>
<thead>
<tr>
<th>BC type</th>
<th>boundary</th>
<th>fixed</th>
<th>free</th>
<th>ITRA</th>
<th>IROT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>simple support</strong></td>
<td>1, 3</td>
<td>( v, w, ru )</td>
<td>( u, rv, rw )</td>
<td>100</td>
<td>011</td>
</tr>
<tr>
<td></td>
<td>2, 4</td>
<td>( u, w, rv )</td>
<td>( v, ru, rw )</td>
<td>010</td>
<td>101</td>
</tr>
<tr>
<td><strong>symmetry</strong></td>
<td>1, 3</td>
<td>( u, rv, rw )</td>
<td>( v, w, ru )</td>
<td>011</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>2, 4</td>
<td>( v, ru, rw )</td>
<td>( u, w, rv )</td>
<td>101</td>
<td>010</td>
</tr>
<tr>
<td><strong>clamped</strong></td>
<td></td>
<td>all dof fixed</td>
<td></td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td><strong>antisymmetry</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

identical to simple support
**P-1 Boundary Conditions—Record 1**

**IBLN(i), i = 1, 4  IBOND**

**IBLN(i)** boundary-condition code for boundary line $i$, $i=1,4$; see Figure 6.6

- 0 – specified on P-2 records
- 1 – simple support
- 2 – clamped
- 3 – unrestrained
- 4 – symmetry
- 5 – antisymmetry
- 6 – compatibility with displacement on some boundary line (see G-1). This is the same as unrestrained, except that the user is informed that the boundary line is junctured in this case.

**WARNING:** It has been reported that under certain conditions STAGS applies erroneous BC at poles when $\text{IBLN}(i) > 0$ on a side which degenerates to a pole. (Please see “M-2 Shell Surface Constants” and Table 6.1 on page 6-17 for conditions where poles are generated.) It is recommended that $\text{IBLN}(i) = 0$ be set for a side which degenerates to a pole, and that BC be defined using P-2 (see Section 6.4 “Boundary Conditions” and Table 6.2). If this is not done, results should be checked carefully to ensure that proper BC have been applied.

**IBOND** boundary conditions for incremental and basic (prebuckling, prestress) displacements are:

- 0 – identical
- 1 – different; may be used only with INDIC = 1 or 4 (B-1)

```
if  (IBLN(i) = 0 for one or more boundaries) then  go to  P-2
elseif (IBOND = 1) then  go to  P-3
else  follow instructions at end of  P-4
```
**P-2 Boundary Conditions—Record 2**

This record is included only if $\text{IBLN}(i) = 0$ (P-1) for one or more shell-unit boundaries. A single P-2 record is required for each boundary line for which $\text{IBLN}(i) = 0$ has been specified. These P-2 records are ordered according to the boundary-line numbering scheme, 1–4, as shown in Figure 6.6 on page 6-58.

---

**ITRA  IROT**

**ITRA**

a 3-digit binary integer indicating freedom or constraint for each of the three translational displacement components ($u, v, w$). The significance of each binary digit is as follows:

- 0—fixed
- 1—free

The most significant digit corresponds to $u$, and the least significant to $w$.

For example, to specify $u$ fixed and $v, w$ free

$$\text{ITRA} = 011$$

**IROT**

a 3-digit binary integer indicating freedom or constraint for each of the three rotational displacement components ($ru, rv, rw$). For example, to specify $rv$ fixed, and $ru, rw$ free

$$\text{IROT} = 101$$

For the examples given above ($u$, $rv$ fixed; $v, w, ru, rw$ free), the P-2 record for the corresponding boundary would be defined as

$$011 \ 101 \ \$ \ \text{P-2}$$

Note that imbedded spaces are not permitted within either **ITRA** or **IROT**; these two 3-digit binary integers must be separated by one or more spaces or by other legal data terminators (see 5.3 “Input Format Conventions” on page 5-4).

Also note that a separate P-2 record is defined for each additional boundary $i$, if any, indicated by $\text{IBLN}(i) = 0$ (P-1).

\*[IBOND (P-1) basic/incremental BC flag]

if\ (IBOND = 1) then\ go to\ P-3
else\ follow\ instructions\ at\ end\ of\ P-4
**P-3 Boundary Conditions—Record 3**

This record is included only if incremental displacements and displacements in the basic stress state are subject to different constraints (\(\text{IBOND} = 1\), P-1).

---

\[ JBLN(i), i = 1, 4 \]

**JBLN(i)**

boundary-condition code for boundary line \(i, i=1,4\); see Figure 6.6

- \(0\) – specified on P-4 records
- \(1\) – simple support
- \(2\) – clamped
- \(3\) – unrestrained
- \(4\) – symmetry
- \(5\) – antisymmetry
- \(6\) – compatibility with displacement on some boundary line (see G-1).

\[ \text{if } (JBLN(i) = 0 \text{ for one or more boundaries) then go to P-4} \]
\[ \text{else follow instructions at end of P-4} \]
P-4 Boundary Conditions—Record 4

This record is included only if \( JBLN(i) = 0 \) (P-3) for one or more shell-unit boundaries. A P-4 record is defined for each boundary line where \( JBLN(i) = 0 \) has been specified. P-4 records are ordered according to the boundary-line numbering scheme, 1 – 4, as shown in Figure 6.6.

\[
\begin{align*}
\text{JTRA} & \quad \text{JROT} \\
\text{JTRA} & \quad \text{a 3-digit binary integer indicating freedom or constraint for each of the three translational displacement components} \ (u, v, w). \ \text{See} \ \text{ITRA} \ (P-2). \\
\text{JROT} & \quad \text{a 3-digit binary integer indicating freedom or constraint for each of the three rotational displacement components} \ (ru, rv, rw). \ \text{See} \ \text{IROT} \ (P-2).
\end{align*}
\]

goto Q-1
6.5 Loads

STAGS utilizes two independent load systems, load system A and load system B. Loading on the structure is computed as

\[ p = P_A \cdot f_A + P_B \cdot f_B \]

where

- \( P_A \) and \( P_B \) are load factors for load systems A and B.
- \( f_A \) and \( f_B \) are base loads for load systems A and B.
- \( P_A \cdot f_A \) and \( P_B \cdot f_B \) are scaled loads for load systems A and B.
- \( p \) represents the applied loads.

Load factors, \( P_A = PA \) and \( P_B = PB \), are specified on the C-1 record (in the case.bin file) for static analysis and on the E-3/E-4 records for dynamic analysis. User-written subroutine FORCET allows a more generalized load-factor history for transient analysis. For shell units, base loads \( f_A \) and \( f_B \) are defined on the Q-1–Q-3 records. Base loads can also be specified via: U-1–U-3 records, for element units; user-written subroutine USRLD, which is an alternative to the Q/U records; and user-written subroutine TEMP, for thermal loading. User-written subroutine UPRESS provides a more generalized pressure-loading feature. Inertial loads defined by the B-6 records are also included in the base load vectors and are scaled by the appropriate load factors.

In addition to forces, base loads include prescribed displacements. We distinguish between prescribed displacements and initial conditions, both of which are defined on Q-3 when \( LT = -1 \). Prescribed displacements are nonzero displacement constraints which may be specified in both static and dynamic analysis. Initial conditions represent nodal displacements and velocities applied to the structure at time \( t = TMIN \) (E-1) in a transient analysis. Initial displacements are equivalent to statically-applied prescribed displacements, with the associated constraints removed upon commencement of the transient analysis \( (t > TMIN) \). Initial conditions are meaningful only when \( INDIC = 6 \) or 7 (B-1), and for initial velocities, when \( IVELO = 1 \) (E-2).

It follows that prescribed displacements and initial conditions are mutually exclusive events. Where a nodal degree-of-freedom (\( dof \)) has been constrained (either fixed or prescribed nonzero), an initial condition applied to that \( dof \) is ignored, since it no longer is present in the system of equations due to the specified constraint. Similarly, where a \( dof \) has been fixed by a
boundary condition (P records), a prescribed displacement is ignored. Notice the order of precedence where conflicting conditions have been specified: fixity takes precedence over a prescribed displacement, which takes precedence over an initial condition.

For bifurcation buckling, the critical load combination is given by \( \lambda P_A \cdot f_A + P_B \cdot f_B \), where \( \lambda \) is the computed eigenvalue. That is, the B-system represents a set of fixed loads.

Point forces, line loads, and surface tractions are generally assumed to maintain their original direction during loading. The only exception to this is if a normal pressure (on the Q-3 record) is defined as a “live pressure load.” In that case the pressure is applied normal to the deformed shell surface. If the loads thus are allowed to rotate during deformation the system is not necessarily conservative.

If IUPLDA or IUPLDB (ET-1) is set to unity, the load sets are assumed to rotate with the nodes in question (follower loading). This option can be useful when modeling an endplate with boundary conditions and line loads, since ordinary line loads do not account for the rotation of the plate. Since these loads are not conservative, they should be used with care.

If a nonzero displacement is defined on a Q-3 record, this displacement will be multiplied by the load factor in the same way as a regular load. The option can also be used for introduction of constraints, as discussed under Section 6.4 “Boundary Conditions”. Fixing a displacement to zero introduces a constraint.

The basic loads with fixed direction in any of the load systems can be defined either partially or totally by a user-written subroutine USRLD. A live load that varies with the shell coordinates must be defined in a user-written subroutine UPRESS. In contrast to other forms of loading, the distribution of the live pressures defined in UPRESS can vary with time.

A uniform stress state is defined for bifurcation buckling or vibration analysis in a Q-5 record. Load system records in that case are only meaningful if they apply homogeneous displacement constraints.

As was described under Section 6.4 “Boundary Conditions”, it is possible to specify incremental boundary conditions for a bifurcation analysis (INDIC = 1 or 4, B-1) that can be different from those for the pre-critical stress state. For shell units, if IBOND = 1 (P-1), specified displacements for load system A apply to the prebuckling state, and specified displacements for load system B become the incremental constraints. It should be noted that only a zero specified displacement
may be used in load B for buckling boundary conditions. It should also be noted that a buckling analysis must be requested to make this meaningful. For element units, set INDIC = −1 (B-1) to achieve the same effect.

The cable hinge moment and cable hinge restraint are a generalized moment and complementary constraint condition suitable for a large rotation response to moment loading. For linear analysis or when the rotations at a node are small, the cable hinge moment is identical to a torsional moment directed along the axis of a cable, which is usually modeled with standard beam elements like the E210 beam element (see Chapter 14 “The Element Library”). When rotations become large, the torsional moment is no longer conservative and will produce poor convergence and unanticipated response. A conservative hinge moment that is in agreement with what is expected for a hinged cable under torsional moment loading is

\[ m_q = \frac{q + e}{1 + q \cdot e} \left| m_e \right| \]

where \( q \) is the unit vector along the current beam axis at the node being loaded, where \( e \) is the user load direction (the unit vector in the direction of the applied moment), where \( \left| m_e \right| \) is the magnitude of the applied moment, and where \( \left| m_q \right| \) is the point load at the node used by STAGS. The cable hinge constraint is the corresponding boundary condition resulting from applying just the right amount of cable moment loading to prevent axial twist. Its small-rotation counterpart is the suppression of axial rotational freedom.
### Q-1 Loads Summary

This record summarizes the load data to be defined for the shell unit. It is always included in each of the shell units.

Temperatures are not defined on Q records. See the C-1 record in Chapter 11 and user-written subroutine TEMP in Chapter 12.

<table>
<thead>
<tr>
<th>NSYS</th>
<th>NICS</th>
<th>NAMS</th>
<th>NUSS</th>
<th>NHINGE</th>
<th>NMOMNT</th>
<th>NLEAST</th>
<th>IPRESS</th>
</tr>
</thead>
</table>

| NSYS  | number of load systems. There are two load systems available—load system A and load system B. Either or both may be included; \( \vdash \) **NSYS** \( \leq 2 \)
|-------|-------------------------------------------------|
| NICS  | number of sets of initial conditions for a transient analysis (**INDIC** = 6 or 7) (B-1). Two types of initial conditions are permissible—initial velocity and initial displacement. Either or both may be included; \( \vdash \) **NICS** \( \leq 2 \). These conditions are prescribed at time \( t = TMIN \) (E-1). If **INDIC** \( \leq 5 \), initial conditions are irrelevant, though they still may be defined, perhaps in preparation for an anticipated transient analysis.
| NAMS  | number of attached masses
| NUSS  | **NUSS** \( > 0 \) indicates that a uniform basic stress state will be defined on Q-5 for vibration or bifurcation buckling
| NHINGE| number of cable hinge restraint vectors
| NMOMNT| number of cable hinge moment loading vectors (must be 0, currently)
| NLEAST| number of least-squares loading sets
| IPRESS| 0 – no UPRESS
|       | 1 – UPRESS is included

User-written subroutine UPRESS defines pressure—surface loading which acts normal to the element surface. The inclusion of UPRESS precludes the definition of pressure loading in the **INP** file. When **IPRESS** = 1, any pressure loads defined on Q-3/U-3 records (**LT** = 4 or 5) are ignored.
if \((\text{NSYS} > 0) \text{ or } (\text{NICS} > 0))\) then \(\text{go to } Q-2\)

elseif \((\text{NAMS} > 0)\) then \(\text{go to } Q-4\)

elseif \((\text{NUSS} > 0)\) then \(\text{go to } Q-5\)

elseif \((\text{NHINGE} > 0)\) then \(\text{go to } Q-6\)

elseif \((\text{NMOMNT} > 0)\) then \(\text{go to } Q-7\)

elseif \((\text{NLEAST} > 0)\) then \(\text{go to } Q-8a\)

else \(\text{go to } R-1\)
**Q-2 Load Set Summary**

The number of loadsets, \( S_l \), is defined as \( S_l = \text{NSYS} + \text{NICS} \) (Q-1), where \( \text{NSYS} \leq 2, \text{NICS} \leq 2 \); therefore \( S_l \leq 4 \). Each of the \( S_l \) load sets is defined in a Q-2/3 series. The order in which they are defined is established as:

- load system A
- load system B
- initial displacement
- initial velocity

All four load sets are optional. Each load set which is present must be completely defined via a Q-2/3 series before moving on to the next set in the order specified above.

<table>
<thead>
<tr>
<th>ISYS</th>
<th>NN</th>
<th>IFLG</th>
</tr>
</thead>
</table>

for load systems:

1 - load system A
2 - load system B

for initial conditions:

0 - initial displacements
-1 - initial velocities

\( \text{NN} \) number of Q-3 records needed to describe the loads or initial conditions

\( \text{IFLG} \) 0 - no USRLD
1 - loads defined in user-written subroutine USRLD are included

if \((\text{NN} > 0)\) then *go to Q-3*
else *follow instructions at end of Q-3*
Q-3 Load Definition

This record is included only if \texttt{NN} > 0 (Q-2); it is repeated \texttt{NN} times. A single Q-3 record is typically used as described in the following three paragraphs to specify loading(s) or displacement(s) at a given point of, along a given row or column of, or over the entire grid of the current shell unit. A convenient \emph{exterior-looping} parameter \texttt{(NX)} can also be used to make multiple such definitions with the same Q-3 record, as described after that.

When the row number is entered as zero, the load (or initial condition) is assumed to act at every grid point on the column indicated by the column number. If the column number is entered as zero, the load is assumed to act at every grid point on the row indicated by the row number. If both row and column numbers are zero, then the load is assumed to act at each grid point.

The unknowns at a juncture line are represented by the displacement components in the directions of the shell coordinates $(X', Y', Z')$ associated with the shell unit having the lowest number of those involved in the juncture. This must be observed if displacement constraints are introduced by Q-2 records, by Q-3 records or in user-written UCONST or USRLD subroutines at a juncture line. In general this is most readily achieved by introduction of the constraint on the shell unit with the lowest number. Some care must be exercised when regular boundary conditions (P-1–P-4) are introduced on the lines that meet the juncture line.

Line loads can be applied as uniform along an entire row or column, or they can be applied as constant on a single edge. Refer to Figure 6.7 on page 6-73 for examples of how to specify edges for line loads. To apply a non-uniform variation along a given line, define a separate load value for each edge on that line. Best results will be obtained by specifying centroidal values for each edge on the line.

As noted above, the optional \texttt{NX} parameter (and the three incrementation parameters associated with it) may be used in combination with the other information on a Q-3 record to make multiple loading/displacement definitions. This feature can be very useful in situations where similar definitions are required for one or another of the three variables (\texttt{LD}, \texttt{LI}, or \texttt{LJ}) for which FORTRAN-like looping might be appropriate.
Loads Model Input—Shell Units

P  LT  LD  LI  LJ  LAX  NX  INC1  INC2  INC3  ILAY

**P**
magnitude of load, displacement, or velocity; the load type and direction are determined by LT and LD, respectively

**LT**
-1 – prescribed displacement (translation or rotation), or initial condition (initial displacement or initial velocity)
1 – point force/moment
2 – line load/moment along row
3 – line load/moment along column
4 – dead pressure—normal to the undeformed element surface
5 – live pressure—normal to the deformed element surface throughout geometrically-nonlinear deformations
6 – velocity-dependent forces (see E-1) — INACTIVE
7 – surface traction — INACTIVE

**LD**
load direction; interpretation of \((X, Y, Z)\) is dependent upon LAX, below
1 – \(u\); force/translation in the \(X\) direction
2 – \(v\); force/translation in the \(Y\) direction
3 – \(w\); force/translation in the \(Z\) direction
4 – \(ru\); moment/rotation, right-handed about the \(X\) axis
5 – \(rv\); moment/rotation, right-handed about the \(Y\) axis
6 – \(rw\); moment/rotation, right-handed about the \(Z\) axis

**LI, LJ**
row and column number of the grid points at which \(P\) is applied.
\((LI, LJ = 0, \text{see above.})\)

**LAX**
load axes, determining the interpretation of LD, above
0 – \((X, Y, Z)\) correspond to \((X', Y', Z')\) shell coordinates
1 – \((X, Y, Z)\) correspond to \((x^e, y_g, z_g)\) nodal-global coordinates
2 – \((X, Y, Z)\) correspond to \((x_e, y_e, z_e)\) element-edge coordinates

**LAX** = 0 is required for LT = -1
LAX = 2 is permitted for line loads only
LAX = 2 is permitted for the 410 and 411 elements only
LAX = 2 is permitted only for the quadrilateral plate (ISHELL = 3) or for a user-written shell (ISHELL = 1 and user-written subroutine LAME); this option is not necessary for other shell types, since a computational axis will serve as an edge reference in all other shell types
The \((z')\) element normal defines the direction of pressure

\(\text{NX}\)

*exterior-looping* parameter; set equal to unity by \textit{STAGS} if nonpositive or omitted

\(\text{INC1}\)

incrementation parameter for the \textit{LD} variable, when \(\text{NX} > 1\)

\(\text{INC2}\)

incrementation parameter for the \textit{LI} variable, when \(\text{NX} > 1\)

\(\text{INC3}\)

incrementation parameter for the \textit{LJ} variable, when \(\text{NX} > 1\)

\(\text{ILAY}\)

layer- or surface-specification parameter; for non-solid shells, \textit{ILAY} is ignored; for solid shells, \textit{ILAY} designates the *nodal layer* on which the specified *displacement*, *line* or *point loading* is to be applied—or it designates the *element layer* to which the specified *pressure loading* is to be applied

\(\text{NAMS}\) (Q-1) number of attached masses

\(\text{NUSS}\) (Q-1) uniform basic stress state option for eigenanalysis

\(\text{NHINGE}\) (Q-1) number of cable hinge restraint vectors

\(\text{NMOMNT}\) (Q-1) number of cable hinge moment loading vectors

\(\text{NLEAST}\) (Q-1) number of least-squares loading sets

\[
\begin{align*}
\text{if} & \quad (\text{NAMS} > 0) \quad \text{then} \quad \text{go to Q-4} \\
\text{elseif} & \quad (\text{NUSS} > 0) \quad \text{then} \quad \text{go to Q-5} \\
\text{elseif} & \quad (\text{NHINGE} > 0) \quad \text{then} \quad \text{go to Q-6} \\
\text{elseif} & \quad (\text{NMOMNT} > 0) \quad \text{then} \quad \text{go to Q-7} \\
\text{elseif} & \quad (\text{NLEAST} > 0) \quad \text{then} \quad \text{go to Q-8a} \\
\text{else} & \quad \text{go to R-1}
\end{align*}
\]
Figure 6.7  Shell-unit line loads; examples of how to specify element edges.
**Q-4 Attached Mass**

The Q-4 records allow the user to define attached masses at node points in the shell unit. The record is repeated NAMS (Q-1) times.

<table>
<thead>
<tr>
<th>GM</th>
<th>weight of attached mass, in units of force</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRM</td>
<td>row number of attached mass</td>
</tr>
<tr>
<td>ICM</td>
<td>column number of attached mass</td>
</tr>
<tr>
<td>LAYER</td>
<td>nodal layer number for attached mass</td>
</tr>
<tr>
<td>NX</td>
<td>number of masses to be attached</td>
</tr>
<tr>
<td>INC1</td>
<td>incrementation parameter for the IRM variable</td>
</tr>
<tr>
<td>INC2</td>
<td>incrementation parameter for the ICM variable</td>
</tr>
<tr>
<td>INC3</td>
<td>incrementation parameter for the LAYER variable</td>
</tr>
</tbody>
</table>

**NUSS** (Q-1) uniform basic stress state option for eigenanalysis

**NHINGE** (Q-1) number of cable hinge restraint vectors

**NMOMNT** (Q-1) number of cable hinge moment loading vectors

**NLEAST** (Q-1) number of least-squares loading sets

if (NAMS attached-mass records have been defined) then
  if (NUSS > 0) then go to Q-5
  elseif (NHINGE > 0) then go to Q-6
  elseif (NMOMNT > 0) then go to Q-7
  elseif (NLEAST > 0) then go to Q-8a
  else go to R-1
else continue defining Q-4
**Q-5 Uniform Stress State for Eigenanalysis**

This record may be used to define the stress resultants in a uniform basic stress state for bifurcation buckling or small vibration analysis. When a Q-5 record is included, the only type of loading allowed is specified homogeneous displacements (\(LT = -1, Q-3\)). Any other loadings (mechanical or thermal) which may be defined are ignored, not only in the current unit, but in all units. The stress resultants defined on this record are multiplied by the load factors \(STLD(1), STLD(2)\) (C-1) for load systems A and B, respectively.

| PNXA | value of the stress resultant in the \(x'\) direction, load system A |
| PNYA | value of the stress resultant in the \(y'\) direction, load system A |
| PNXYA | value of the shear resultant, load system A |
| PNXB | value of the stress resultant in the \(x'\) direction, load system B |
| PNYB | value of the stress resultant in the \(y'\) direction, load system B |
| PNXYB | value of the shear resultant, load system B |

\[
\begin{align*}
\text{NHINGE} &\quad (Q-1) \quad \text{number of cable hinge restraint vectors} \\
\text{NMOMNT} &\quad (Q-1) \quad \text{number of cable hinge moment loading vectors} \\
\text{NLEAST} &\quad (Q-1) \quad \text{number of least-squares loading sets}
\end{align*}
\]

if \((\text{NHINGE} > 0)\) then \(\text{go to Q-6}\)

elseif \((\text{NMOMNT} > 0)\) then \(\text{go to Q-7}\)

elseif \((\text{NLEAST} > 0)\) then \(\text{go to Q-8a}\)

else \(\text{go to R-1}\)
Q-6 Cable Hinge Restraint

This record is included only if NHINGE > 0 (Q-1); it is repeated NHINGE times.

See discussion of cable hinge restraints under Q-1. The vector described below gives the direction of the constraint in (X', Y', Z') shell coordinates. Although the magnitude of this vector does not theoretically influence the result, this is a multi-point constraint imposed by Lagrange multipliers. The new equation (automatically) produced by this constraint is assembled into the stiffness matrix along with the other freedoms. To avoid numerical problems, the magnitude of this vector is converted to a scaling constant to help the conditioning of the stiffness matrix. See the general discussion of Lagrange constraints under G-3 for more details.

<table>
<thead>
<tr>
<th>IROW</th>
<th>ICOL</th>
<th>HRU</th>
<th>HRV</th>
<th>HRW</th>
<th>LAYER</th>
<th>NX</th>
<th>INC1</th>
<th>INC2</th>
<th>INC3</th>
</tr>
</thead>
<tbody>
<tr>
<td>IROW</td>
<td>row number where constraint is applied</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICOL</td>
<td>column number where constraint is applied</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HRU, HRV, HRW</td>
<td>the direction vector, in (X', Y', Z') shell coordinates, along which rotation is prevented; the magnitude of this vector should be of the same order as the element torsional stiffness</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAYER</td>
<td>nodal layer on which constraint is applied</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NX</td>
<td>number of constraints to be attached</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC1</td>
<td>incrementation parameter for the IROW variable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC2</td>
<td>incrementation parameter for the ICOL variable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC3</td>
<td>incrementation parameter for the LAYER variable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NHINGE (Q-1) number of cable hinge restraint vectors
NMOMNT (Q-1) number of cable hinge moment loading vectors
NLEAST (Q-1) number of least-squares loading sets

if (NHINGE cable hinge restraint vectors have been defined) then
  if (NMOMNT > 0) then go to Q-7
  elseif (NLEAST > 0) then go to Q-8a
  else go to R-1
else continue defining Q-6
Q-7 Cable Hinge Moment

NOTE: This feature is currently disabled, and the user must set \( \text{NMOMNT} = 0 \) (Q-1). \textit{STAGS} automatically introduces cable moments for all applied moment loadings.

This record is included only if \( \text{NMOMNT} > 0 \) (Q-1); it is repeated \( \text{NMOMNT} \) times.

See discussion of cable hinge moment loads under Q-1. The vector described below gives the direction and magnitude of the applied moment in \((X', Y', Z')\) shell coordinates.

<table>
<thead>
<tr>
<th>IROW</th>
<th>ICOL</th>
<th>MSYS</th>
<th>RUM</th>
<th>RVM</th>
<th>RWM</th>
<th>LAYER</th>
<th>NX</th>
<th>INC1</th>
<th>INC2</th>
<th>INC3</th>
</tr>
</thead>
<tbody>
<tr>
<td>IROW</td>
<td>row number where moment is applied</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICOL</td>
<td>column number where moment is applied</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSYS</td>
<td>1 – load applies to the A system 2 – load applies to the B system</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RUM, RVM, RWM</td>
<td>moment vector components in ((X', Y', Z')) shell coordinates</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAYER</td>
<td>nodal layer on which moment is applied</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NX</td>
<td>number of moments to be applied</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC1</td>
<td>incrementation parameter for the \textit{IROW} variable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC2</td>
<td>incrementation parameter for the \textit{ICOL} variable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC3</td>
<td>incrementation parameter for the \textit{LAYER} variable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( \text{NMOMNT} \) (Q-1) number of cable hinge moment loading vectors  
\( \text{NLEAST} \) (Q-1) number of least-squares loading sets

if \( \text{NMOMNT} \) (cable hinge moment loading vectors have been defined) then  
if \( \text{NLEAST} > 0 \) then \textit{go to Q-8a}  
else \textit{go to R-1}  
else \textit{continue defining Q-7}
**Q-8a Least Squares Loading Summary**

There are \( \text{NLEAST} (Q-1) \) sets of Q-8 records. Record Q-8a is used to define the number \( \text{NSQR} \) of Q-8b special loading records to follow, to define the reference node for the application of the least-squares summary loads or reactions, and to provide an overall scale factor \( \text{SCALE} \) for the six Lagrange constraints introduced automatically for each least-squares load set. The reference node must have been defined in the current or a previous unit. If any nodes not yet defined are referenced in any Q-8 records, an error will result; to avoid this, postpone the definition of this least-squares set until after all pertinent nodes have been defined. As with all other Lagrange constraints that contribute to the stiffness matrix (see, for example, record G–3 and the ones that follow it), numerical roundoff considerations require that the values of the Lagrange unknowns not differ from other unknowns by too many orders of magnitude. This in turn means that the stiffness contributions should be near the same order as other members, or a \( \text{SCALE} \) value set to about equal to some average thickness times a modulus. The user should remember that this is only an order-of-magnitude estimate for numerical conditioning.

<table>
<thead>
<tr>
<th>NSQR</th>
<th>IUNIT</th>
<th>IROW</th>
<th>ICOL</th>
<th>SCALE</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{NSQR} )</td>
<td>number of Q-8b data records required to define the edges involved in this least-squares load set</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{IUNIT} )</td>
<td>unit number for the reference node</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{IROW} )</td>
<td>row number for the reference node</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{ICOL} )</td>
<td>column number for the reference node</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{SCALE} )</td>
<td>scale factor for numerical conditioning. We suggest something of the order of magnitude of the material modulus times a representative shell thickness.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*go to Q-8b*
Q-8b Least Squares Load Definition

A least-squares constraint loading condition is specified by defining the following three components:

- A reference node with section weighting (see Q-8a)
- A curve consisting of one or more arc segments and/or single nodes (Q-8b).
- Loads on the reference node (see Q-1, Q-2, Q-3 records)

Q-8b input records define a curve consisting of one or more arc segments, or alternatively, individual nodes. There are a total of \( NSQRS \) Q-8b records. A sequence of Q-8b records, all with \((LU, LR, LC)\) defining actual nodes, specifies one arc segment. Additional arc segments may then be defined by adding a Q-8b record with \((LU, LR, LC) = 0\) followed by more Q-8b records defining the next arc segment. If a segment consists of a single node, the node weight \( P \) is used unchanged. An individual Q-8b record can specify several nodes at once in two ways:

- For shell units, \( LR = 0 \) and \( LC > 0 \) means the entire column \( LC \), while \( LR > 0 \) and \( LC = 0 \) means the entire row \( LR \).
- For either shell or element units, items \( LNDA \), \( LNDB \), and \( LNDINC \) may be used to describe a range of nodes.

\[
\begin{array}{cccccc}
P & LU & LR & LC & LNDA & LNDB & LNDINC \\
\end{array}
\]

- \( P \): nodal weight. For sections of uniform composition, input 1.0. When the defining curve belongs to several distinct wall types, we suggest using 1.0 for the nodes in the stiffest structural areas and choosing other weights so that the set of \( P \) values are proportional to the modulus times the effective thickness (\( Eh \)).
- \( LU \): unit number of node; any Q-8b records that follow describe a new arc segment to be added to this load set
- \( LR \): row number of first node
- \( LC \): column number of first node
- \( LNDA \): first node of (row if \( LR = 0 \)) or (column if \( LC = 0 \)) in shell unit. If 0, the first node on the row or column is chosen.
LNDB  last node of (row if \( LR = 0 \)) or (column if \( LC = 0 \)) in shell unit. If 0, the last node on the row or column is chosen. If \( LNDB = LNDA > 0 \), then only one node is specified by this record.

LNDINC  increment for nodes from LNDA to LNDB. Increasing or decreasing ranges are allowed. If LNDINC = 0, an increment of +1 is used for increasing ranges, and -1 for decreasing ranges.

NSQR  (Q-8a)  number of Q-8b records
NLEAST  (Q-1)  number of least-squares loading sets

if  (NSQR Q-8b records have been defined)then
  if  (NLEAST loading sets have been defined)then  go to  R-1
  else  return to  Q-8a
else  continue defining  Q-8b
6.6 Least-Squares Distributed Line Loads

Definition of least-squares constraint

Users are often faced with the situation in which a detailed response is required in only part of a structure, such as that shown by the shaded portion of the airplane in Figure 6.7 “Shell-unit line loads; examples of how to specify element edges.” on page 6-73. For the rest of the structure, an approximate analysis may be sufficient to provide enough information to define the load path along the boundaries of the area of interest, shown in the lower half of this figure. A very good estimate of the integrated boundary loads can sometimes be computed with simple “stick models” consisting of beams with the appropriate section properties. Unfortunately, with this approach the detailed edge loading (arrows in the figure) is lost. It is time-consuming and error-prone to try to reconstitute these loads by hand; and even when a good approximate initial estimate can be computed, changes in the load path as a function of edge deformation cannot be accounted for. This is a more serious problem when only engineering statically-equivalent forces...
and moments acting on the section centroid are available, or when these loads come from unknown reactions generated by an approximate combined model. How, then, can one compute a “best” estimate for the edge loading automatically, one that induces the least spurious deflection, when only the integrated edge force and moment pair acting on the centroid are available?

One plausible solution to this problem is to recognize that the virtual work done by a force $F$ and moment $M$ directed at a reference node—a point in the structure (presumably the centroid of the section, marked by a “+” in the lower half of Figure 6.8)—is

$$\delta W = F\delta u_0 + M\delta \omega_0$$

(6.1)

where $\delta u_0$ is the virtual displacement, and $\delta \omega_0$ is the virtual instantaneous rotation. In rigorous language, this means that the force/moment pair is conjugate to the rigid translation and rotation of the edge, and that the work done by all of the edge loads is equivalent to equation (6.1), regardless of how the load is distributed over the edge. At this stage of the modeling process, we will have at hand a finite element discretization of the detailed section, with a set of nodes lying on the edge, as illustrated by the dots in the following figure. Imagine that initially we have
defined a stretch of nodes that are to receive the loads produced by \( \mathbf{F} \) and \( \mathbf{M} \). The minimum disturbance of the edge is produced by requiring a least-squares error between the actual deflection of the section node and the undeformed section rotating as a rigid body and doing equivalent work on \( \mathbf{F} \) and \( \mathbf{M} \). We illustrate this in Figure 6.8. On the left side of the figure, we show an undeformed cross section with a number of open circles with dots. Each circle represents a node along the loaded edge, and the + mark represents the loaded reference node. The reference node is an arbitrary structural node, which can be loaded or connected to other parts of the structure. Initially, the undeformed structure and the loaded section are coincident. Later, the structure deforms as shown on the right side of the figure. Here, we have shown the section ovalizing and carrying its nodes with it (dots). We also show an image of the undeformed section that has translated and rotated to make the square of the distances \( \mathbf{d}_i \) between the deformed node and its rotated undeformed image (open circle) a minimum. One of these deflections (arrows pointing from the image to the deformed node) for node \( i \) is shown in a box, labeled with its defining equation

\[
\mathbf{d}_i = \mathbf{r}_i - \Delta \mathbf{x}_i
\]

\[
\mathbf{r}_i = \mathbf{T}_0 \mathbf{r}_i^0
\]

\[
\mathbf{r}_i^0 = \mathbf{x}_i - \mathbf{x}_0
\]

\[
\Delta \mathbf{x}_i = \mathbf{x}_i - \mathbf{x}_0
\]

where \( \mathbf{x}_i \) are the undeformed coordinates of node \( i \), where \( \mathbf{0} \) refers to the reference node, and where \( \mathbf{x}_i \) are the nodal positions after deformation by displacements \( \mathbf{u}_i \), (including the reference node):

\[
\mathbf{x}_i = \mathbf{x}_i + \mathbf{u}_i
\]

These vectors can be visualized with the help of the following figure. The orthogonal matrix \( \mathbf{T}_0 \) represents the rigid rotation of the reference node and the section image tied to it. What we want is to rotate and translate the image section to make the least-squares error \( \varepsilon \)

\[
\varepsilon = \sum_i \phi_i \| \mathbf{d}_i \|^2
\]
a minimum. If we take the derivative of equation (6.4) and remember that the variation of a vector subjected to a rigid rotation (the second equation in (6.2)) is

$$
\delta \mathbf{r}_i = \delta \omega_0 \times \mathbf{r}_i
$$

(6.5)

the result is six constraint equations expressed as

$$
\sum_i (\mathbf{r}_i - \Delta \mathbf{x}_i) \phi_i = 0
$$

$$
-\sum_i (\mathbf{r}_i \times \Delta \mathbf{x}_i) \phi_i = 0
$$

(6.6)

where $\phi_i$ are weights chosen by the user. Usually these weights are selected to be proportional to the length of section segments spanned by the nodes. The most straightforward way to introduce these constraints into a finite element code like STAGS is to augment the strain energy function $U$ by introducing Lagrange multipliers $\mathbf{f}_0$ and $\mathbf{m}_0$ as follows:

$$
\hat{U} = U + \mathbf{f}_0 \cdot \left[ \sum_i (\mathbf{r}_i - \Delta \mathbf{x}_i) \right] - \mathbf{m}_0 \cdot \left[ \sum_i \mathbf{r}_i \times \Delta \mathbf{x}_i \right]
$$

(6.7)
The first variation

To obtain contributions to the residual, we take the variation of equation (6.7), the results of which come out to be for each node $i$

$$
f_i = \begin{bmatrix}
  r_i \times m_0 - f_0 \\
  f_0 - r_i \times m_0 \\
  m_0(\Delta x_i \cdot r_i) - \Delta x_i (r_i \cdot m_0) + r_i \times f_0 \\
  r_i - \Delta x_i \\
  -r_i \times \Delta x_i
\end{bmatrix} \phi_i
$$

(6.8)

where the order of the freedoms is $u, v, w$ for the structural node $i$; $u, v, w, R_u, R_v, R_w$ for the reference node 0; and the last six for the Lagrange unknowns $f_0$ and $m_0$, respectively. There is a contribution to the reference node and the Lagrange freedoms from each $f_i$, so that one could view the $f_i$ as being generated by an “element,” shown by the dashed line connecting the reference node and a deformed edge node $i$ in Figure 6.9; there is one such “element” connecting each edge node on the section with the reference node, like spokes on a wheel. Also note that there is no contribution to the rotational freedoms at the edge. If we take the simple case of a reference node unconnected in any way to the structure except for these constraints, and if we assume that the node is loaded by a simple force $F$ and moment $M$, then equilibrium requires that

$$
F = w f_0 - \hat{r} \times m_0 \\
M = J_s m_0 + \hat{r} \times f_0
$$

(6.9)

$$
w = \sum_i \phi_i \\
\hat{r} = \sum_i \phi_i r_i \\
J_s = \sum_i \phi_i [I(\Delta x_i \cdot r_i) - \Delta x_i r_i^T]
$$

If we simplify the problem further and consider only the case where the reference node is at the center of the section, such that the relative centroid

$$
\hat{r} = 0
$$

(6.10)
then from the first of equations (6.6), we see that $\mathbf{f}_i$ times the sum of the weights $W$ equals the total force on the section. Indeed, the first constraint in equation (6.6) is simply a statement that the relative centroid remains unchanged, and the second term in the force equation (6.9) is the correction for the offset of the reference point from the actual centroid. $\mathbf{J}_s$ can be interpreted as a generalized area moment, since from equations (6.6) it is a symmetric matrix and is initially equal to the area moment of the undeformed section. Most importantly, the reader can readily verify that

$$
\sum_i \mathbf{f}_i = \sum_i (\mathbf{r}_i \times \mathbf{m}_0 - \mathbf{f}_0)\phi_i = -\mathbf{F}
$$

$$
\sum_i \Delta \mathbf{x}_i \times \mathbf{f}_i = \sum_i \Delta \mathbf{x}_i \times (\mathbf{r}_i \times \mathbf{m}_0 - \mathbf{f}_0)\phi_i = -\mathbf{M}
$$

which is clearly required to guarantee that the virtual work done by the edge nodes is equivalent to the work done by the reference node on $\mathbf{F}$ and $\mathbf{M}$. To prove the second equation in (6.11), we use both constraint conditions (equations (6.6)). Finally, for the simple case of a circular cut through a cylinder, the cross product in the first line of equation (6.8) reduces to the familiar cosine distribution derived from an imposed moment.

**The stiffness matrix**

The second variation is straightforward, so we shall only present the final results here. The lower triangle of the symmetric stiffness matrix $\mathbf{K}_i$ looks like

$$
\mathbf{K}_i = \phi_i \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & -K_{si} & K_{mo} & 0 \\
-K_{fi} & K_{fi} & K_{mi} & 0 \\
-K_{mi} & -K_{mi} & K_{mo} & 0
\end{bmatrix}
$$

(6.12)

where we have ordered the freedoms for each “element” $i$ in the same manner as in equation (6.8), and where $\omega$ refers to the rotational components of the reference node. Each of the nonzero 3x3 blocks is defined as follows:
\[
\begin{align*}
\mathbf{K}_{\text{ml}} &= \mathbf{r}_i \mathbf{m}_0^T - l(\mathbf{r}_i \cdot \mathbf{m}_0) \\
\mathbf{K}_{\text{fr}} &= -l \\
\mathbf{K}_{\text{ml}} &= \tilde{\mathbf{r}}_i \\
\hat{\mathbf{K}}_{\text{w00}} &= \mathbf{m}_0(\mathbf{r}_i \times \Delta \mathbf{x}_i)^T - \Delta \mathbf{x}_i(\mathbf{r}_i \times \mathbf{m}_0)^T + \mathbf{r}_i \mathbf{f}_0^T - l(\mathbf{r}_i \cdot \mathbf{f}_0) \\
\mathbf{K}_{\text{w00}} &= \frac{1}{2}(\hat{\mathbf{K}}_{\text{w00}} + \hat{\mathbf{K}}_{\text{w00}}^T) \\
\mathbf{K}_{\text{m0}} &= l(\Delta \mathbf{x}_i \cdot \mathbf{r}_i) - \Delta \mathbf{x}_i \mathbf{r}_i^T
\end{align*}
\]

where \( \tilde{\mathbf{r}}_i \) represents the skew-symmetric matrix derived from the vector \( \mathbf{r}_i \).

**Using least-squares loading in STAGS**

We are fortunate that in STAGS we are able to use existing software to compute automatically the weights \( \phi_i \) based on the correct internodal distances. The user will be required to provide only a simple overall *scale factor*, and a section *weight factor* to be used to account for changes in thickness or other modeling features he wants to include. As users become accustomed to the least-squares loading feature, their experience will determine optimum weight choice.

STAGS automatically generates a set of local weights equal to the distance between adjacent nodes in a list given by the user. This list, the designation of the reference node, and the section weight factors are provided by a natural extension of the shell-unit \( Q \) and \( U \) records and the element-unit \( U \) and \( V \) records in the STAGS input stream. The reference node must be defined like any other, either as a part of a previously-defined shell unit, or in an element unit. Of course, this means that an element unit is required if the new node is defined by itself. Loads and boundary conditions for the reference node are defined as usual, including the reference node force and moment that the user wants spread over an edge. The same rules apply to any structural component attached to the reference node: treat the node like any other ordinary STAGS node. After all nodes (edge “structural” nodes and the reference node) have been defined, a separate set of \( Q \) records must be included for each least-squares loading condition. Provision is made on these records to reference any previously-defined unit that may contain nodes involved in the load set.
6.7 Output Control

Output for any given shell unit to the text-based output file (*casename.out2*, for example) is controlled by input on the R records for that unit. R records are specified independently for each shell unit and do not have to be the same for all units. It may be desirable, at times, to refrain from printing results during the analysis operations—and to use post-processing utilities to print and/or display results of interest, as appropriate. Options are provided for printing all displacements, strains, stresses and/or stress resultants for all grid points and/or elements for every solution step or at specified step-number intervals (see the R-1 record). The R-2 record allows the user to specify a subset of grid points at which displacements are to be printed. The R-3 record (which is disabled at the current time) allows the user to specify elements for which stresses will be printed at each load or time step.
R-1 Output Control—Record 1

This record is always included for each shell unit. The constants governing the output can differ from one unit to another. Output of results within some or all of the units can be suppressed in favor of output in the post-processor. Unnecessary output should be avoided as it leads to unnecessary expense and increase in size of the text-based output file. This is particularly true for stresses since the computer time involved in stress computations is significant.

All displacement and nodal point force output are displayed in \((X', Y', Z')\) shell coordinates. This means that the output of quantities along a juncture belonging to two or more units may not be the same, because these vectors are expressed in different local systems. As an example, where an annular plate is joined to a cylinder, the \(w\) (radial, or normal) component for the cylinder is the same as the \(u\) component for the plate. Similarly, the \(w\) component for the plate is in the plus or minus \(u\) direction for the cylinder.

Nodal point equilibrium forces are available and may be printed. These internal forces should balance any loads or reactions applied at the node. They can be used to check how well equilibrium is maintained throughout the structural model because internal forces should be zero everywhere except at points of support or mechanical loading. Since the live pressure load contribution is not included in point force output, it is necessary to estimate the value of the pressure when checking for system equilibrium. On shell unit junctures the point forces include the reactions from the adjacent shell units.

Stress, strain, and stress resultant output are in \((\tilde{x}, \tilde{y})\) fabrication coordinates. Stress output can also be in \((\phi_1, \phi_2)\) material coordinates for the layer in question. (See Section 4.1 “Coordinate Systems” on page 4-1 for a description of all STAGS coordinate systems.) In principle, for a composite, the stresses on the lower and upper layers can be in different systems; the user therefore should be very careful!

It is possible in nonlinear static or in transient analysis to print displacement components at all load or time steps for selected points, rows, or columns. The total number of records defining selected output in all units cannot exceed 100 times the total number of units.
**Model Input—Shell Units**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPRD</td>
<td>0 – do not print displacements&lt;br&gt;&gt;0 – displacements are printed at every IPRD&lt;sup&gt;th&lt;/sup&gt; load or time step</td>
</tr>
<tr>
<td>IPRR</td>
<td>0 – do not print stress resultants&lt;br&gt;&gt;0 – stress resultants are printed at every IPRR&lt;sup&gt;th&lt;/sup&gt; load or time step (elastic analysis only)</td>
</tr>
<tr>
<td>IPRE</td>
<td>0 – do not print strains&lt;br&gt;&gt;0 – strains are printed at every IPRE&lt;sup&gt;th&lt;/sup&gt; load or time step</td>
</tr>
<tr>
<td>IPRS</td>
<td>0 – do not print stresses&lt;br&gt;&gt;0 – stresses are printed at every IPRS&lt;sup&gt;th&lt;/sup&gt; load or time step</td>
</tr>
<tr>
<td>IPRP</td>
<td>0 – no additional stress output for points with yield&lt;br&gt;&gt;0 – stresses and strains are printed at all points, with yield at every IPRP&lt;sup&gt;th&lt;/sup&gt; load or time step</td>
</tr>
<tr>
<td>IPRF</td>
<td>0 – do not print nodal point forces (internal force vector)&lt;br&gt;&gt;0 – nodal point forces are printed at every IPRF&lt;sup&gt;th&lt;/sup&gt; load or time step</td>
</tr>
<tr>
<td>NSELD</td>
<td>number of records defining selected displacements (one record may correspond to a node, a row, or a column)</td>
</tr>
<tr>
<td>NSELS</td>
<td>number of records defining selected stresses (disabled, set NSELS=0)</td>
</tr>
<tr>
<td>IPRDSP</td>
<td>0 – print selected displacements at every load or time step&lt;br&gt;&gt;0 – print selected displacements at every IPRDSP&lt;sup&gt;th&lt;/sup&gt; load or time step</td>
</tr>
<tr>
<td>IPRSTR</td>
<td>0 – print selected stresses at every load or time step&lt;br&gt;&gt;0 – print selected stresses at every IPRSTR&lt;sup&gt;th&lt;/sup&gt; load or time step (disabled)</td>
</tr>
<tr>
<td>ISL</td>
<td>0 – element results are computed at centroids&lt;br&gt;1 – element results are computed at integration points</td>
</tr>
</tbody>
</table>

*Stress output is obtained only where indicated for the corresponding shell wall or beam cross-section. Please check LSOL (K-2), NSOYZ (J-1), ISP (J-3A), and ISOC (J-3B). Also, see ISL, ISS, and ISD, below.*
**ISS**

0  –  no transverse shear stresses

1  –  compute transverse shear stresses — INACTIVE

**ISD**

0  –  print stress and strain components in \((\tau, \gamma)\) fabrication coordinates

1  –  print stress components in both \((\tau, \gamma)\) fabrication coordinates and \((\phi_1, \phi_2)\) material coordinates

2  –  print stress components in both \((\tau, \gamma)\) fabrication coordinates and the principal directions (includes angle of orientation)

---

**NUNITS** (B-2)  number of shell units

**NUNIT** (B-2)  number of element units

if \((\text{NSELD} > 0)\) then  go to R-2

elseif \((\text{NSELS} > 0)\) then  go to R-3

elseif \((\text{NUNITS} \text{ shell units have been defined})\) then

if \((\text{NUNIT} = 0)\) then  data deck is complete

else  follow instructions at end of R-3

else  return to M-1
R-2 Output Control—Record 2

This record defines a number of nodes at which displacements are to be printed at each load or time step.

If IROWD is set equal to zero the displacements are printed for each row, (i.e., the entire column). If ICOLD is set equal to zero the displacements are printed for each column, (i.e., the entire row).

IROWD  ICOLD

IROWD  row number at which displacements are to be printed
ICOLD  column number at which displacements are to be printed

Note: This selected displacement output option also causes the internal forces for the selected notes to be computed and printed. These forces are also summed. This provides a convenient mechanism for printing the total applied load in cases where only applied displacements are specified, or to check equilibrium at boundaries.

NSELD (R-1)  number of selected displacements
NSELS (R-1)  number of selected stresses
NUNITS (B-2)  number of shell units
NUNITE (B-2)  number of element units

if (NSELD R-2 records have been defined) then
  if (NSELS > 0) then go to R-3
  elseif (NUNITS shell units have been defined) then
    if (NUNITE = 0) then data deck is complete
    else follow instructions at end of R-3
  else return to M-1
else continue defining R-2
**R-3 Output Control—Record 3**

**INACTIVE**

This record defines a number of locations at which stresses will be printed at each load or time step. Stress output will be obtained in the directions defined by \texttt{ISD} (R-1) or in user-written subroutines. Stresses suppressed in the general output by \texttt{LSOL} (K-2) or in user-written subroutines cannot be printed as selected output. The stresses in a given element in the shell unit are referred to by use of the lowest values of the numbers of the rows and columns that bound the element. If selected stress output is requested, all stress components are printed at each selected stress output point.

The record is repeated \texttt{NSELS} times (R-1).

---

**IROWS ICOLS**

IROWS row number identifying the element for stress output
ICOLS column number identifying the element for stress output

- \texttt{NSELS} (R-1) number of selected stresses
- \texttt{NUNITS} (B-2) number of shell units
- \texttt{NUNITE} (B-2) number of element units
- \texttt{NUPT} (H-1) number of user points
- \texttt{NT1} (H-1) number of “spring” elements
- \texttt{NT2} (H-1) number of “beam” elements
- \texttt{NT3} (H-1) number of triangular shell elements
- \texttt{NT4} (H-1) number of quadrilateral shell elements
- \texttt{NT5} (H-1) other-elements/element-command-mode flag

if (\texttt{NSELS R-3} records have been defined) then
if (\texttt{NUNITS} shell units have been defined) then
  if \((\texttt{NUNITE} = 0)\) then *data deck is complete*
  elseif \((\texttt{NUPT} > 0)\) then go to S-1
  elseif \((\texttt{NUPT} = 0)\) then go to S-3
  elseif \((\texttt{NUPT} = -1)\) then go to S-3
  elseif \((\texttt{NT1} > 0)\) then go to T-1
  elseif \((\texttt{NT2} > 0)\) then go to T-2
  elseif \((\texttt{NT3} > 0)\) then go to T-3a
  elseif \((\texttt{NT4} > 0)\) then go to T-4a
  elseif \((\texttt{NT5} > 0)\) then go to T-5
  else \texttt{go to U-1}
else \texttt{return to M-1}
else \texttt{continue defining R-3}
Element units are numbered sequentially, in the order in which they are defined, from \texttt{NUNITS + 1} to \texttt{NUNITS + NUNITE}, where \texttt{NUNITS} is the number of shell units and \texttt{NUNITE} is the number of element units (B-2) in the model.

The group of records in the S–V series describes the element units. They are defined in serial fashion, completely describing one element unit before proceeding to the next. Thus, when \texttt{NUNITE} $\geq 2$, the S–V series is defined for the first element unit, then the series is repeated for the second element unit, and so forth, until all of the \texttt{NUNITE} element units have been defined. The S–V record sets contain information as summarized below:

- \texttt{S} records: Nodes and Lines
- \texttt{T} records: Element specifications
- \texttt{U} records: Loading specifications
- \texttt{V} records: Output control specifications

If the model has externally-generated linear-stiffness contributions, \texttt{W}-type records must be included after the all of the S-V records (for all of the element units) have been given.

\texttt{STAGS} input requirements for the \texttt{S} (nodes and lines) records are described in this chapter. Input requirements for the \texttt{T-x} and \texttt{T-xx} (element-definition) records that are used with the historic \textit{Edef (element-definition)} protocol are described in Chapter 8. Input requirements for the \texttt{T-xxx} (element-definition) records that are used with the newer \textit{Ecom (element-command)} protocol are described in Chapter 9. Input requirements for the \texttt{U} (loadings), \texttt{V} (output-control) and \texttt{W} (linear-stiffness-contribution) records are described in Chapter 10.

\textit{A \texttt{STAGS} model may contain any number of shell units, combined with any number of element units.}
7.1 User Points

Elements in a STAGS element unit are typically connected to node points—some of which may have been defined in one or more shell units and/or in one or more previously-specified element units, and some of which may be unique to that element unit. Nodes that are referenced in any given element unit are called the “user points” for that unit. The current version of STAGS gives the user two protocols for defining user points.

If NUPT is positive on the H-1 record for the current element unit, STAGS expects the user to follow the historic upts user-point-definition protocol and to identify (and/or create) exactly NUPT user points. When this protocol is employed, STAGS attempts to read and process one or more S-1/S-1a (and optionally S-2) user-point-definition record sets—all of which are described in this chapter.

If the NUPT parameter is zero on H-1, STAGS expects the analyst to use the newer user–points protocol to identify (and/or create) one or more user points for the current unit. When this protocol is used, STAGS attempts to read and process one or more S-3/S-3a/S-4 user-point-definition record sets (which are also described in this chapter)—continuing to do this until the analyst tells the program (by setting the user-point-identity parameter to an extraordinarily high value) to stop. With the very first of the record sets for this protocol, the analyst must choose to use either one or the other of the two user-point numbering modes, as described in the following two paragraphs:

If on the very first S-3 record the user assigns a positive value to the user-point number, STAGS will use that value for the initial user point and will expect (require) that all user points be identified explicitly on this and on all subsequent S-3/S-3a record sets. In this case, STAGS determines NUPT by setting it equal to the highest user-point number specified by any S-3/S-3a record set in the sequence.

If on the very first S-3 record the user assigns a zero (or any nonpositive) value to the user-point number, STAGS will set the initial user-point number equal to one and will increment each subsequent user-point number (from the first S-3/S-3a record set and from all subsequent S-3/S-3a record sets) by one. In this case, STAGS ignores all of the user-point numbers that are given by the user and/or calculated with user-specified incrementations on each and every S-3/S-3a record set. STAGS sets NUPT equal to the total number of user points that are defined by this process.

If NUPT=−1 on the H-1 record for the current element unit, no user points are to be defined via S-1 or S-3 record groups for this element unit.

In any event, some (or all) of the user points may alternatively be identified (or defined) in a user-written subroutine USRPT, as described in Chapter 12.
**S-1 User Points (upts protocol)**

The *upts* protocol uses records of this type to establish a nodal point list for the element unit. Additional nodes are defined in user-written USRPT if \( \text{IUWP} = 1 \) (H-1). If the node point coincides with a node on one of the shell units, its location can be defined through reference to the row and column number for that node. Full displacement compatibility at the node between shell unit and element unit will then be enforced. Other node point locations (auxiliary nodes) are defined by use of their coordinates in the global system.

For nodes that coincide with a node on a shell unit, the directions of the freedoms (displacement and rotation vector components) are identical to those on the shell unit at that node. When more than one shell unit belongs to such a node, the components belong to the shell unit with the lowest number. The same rule applies when two different element units share a node: the components belong to the lowest unit. All shell units have a lower number than any element units. Any node that has not been created before in a lower unit is called an auxiliary node. For auxiliary nodes, the user has a choice regarding the freedom pattern. Often constraints are readily defined if the directions of the freedom coincide with the axes of the global system. For more general cases, auxiliary systems can be defined on an S-2 record.

Constraints on displacements or rotations can be introduced on this record (\( \text{IUVW, IRUVW} \)). Attached masses are also permitted. Constraints can also be applied as loads (U-3).

User points may be *defined* in any order, but they must be *numbered* from 1 to \( \text{NUPT} \) (i.e., \( 1 \leq \text{IUP} \leq \text{NUPT} \)). If \( \text{IUS} > 0 \), then \( \text{IRS} \) must be \( > 0 \). If the node belongs to another shell unit (\( \text{IUS} \leq \text{NUUNIT} \), B-2), then \( \text{ICS} \) must be \( > 0 \), giving both the row and column number of the node in the lower unit. If \( \text{IUS} > \text{NUUNIT} \), the current element unit is joined to another element unit with a lower number; in that case \( \text{IRS} \) is the node number belonging to the lower element unit, defined on a previous S-1 record or in the user-written routine USRPT, and \( \text{ICS} = 0 \). If this is element unit \( l \) counted in the order that it is defined, \( \text{IUS} = k \), where \( k = \text{NUUNIT} + l \), where \( l \leq \text{NUUNIT} \) (B-2). If \( \text{IUS} = 0 \), the nodal position must be defined in \((x_g, y_g, z_g)\) global coordinates.

Typically, a single S-1 record (with its associated S-2 record, if required) will be used to define each user point. For some configurations, it is convenient to use the S-1 *looping* feature described below to specify two or more closely-related user points simultaneously. The \( \text{NPTS} \) parameter, on the S-1 record, facilitates that.

In any event, the S-1/S-1a, S-2 sequence must be repeated as many times as necessary to specify \( \text{NUPT} \) (H-1) user points.
<table>
<thead>
<tr>
<th>IUPT</th>
<th>IUS</th>
<th>IRS</th>
<th>ICS</th>
<th>XG</th>
<th>YG</th>
<th>ZG</th>
<th>IUVW</th>
<th>IRUVW</th>
<th>IAUX</th>
<th>NPTS</th>
<th>ILAY</th>
</tr>
</thead>
</table>

**IUPT**
user point (node) number; \(1 \leq IUPT \leq NUPT\)

**IUS, IRS, ICS**
shell unit number, row number, column number for node on a shell unit

If \(IUS > NUNIT\), \(IUS = k = NUNIT + l\), where \(l\) is a previously-defined element unit (counted in the order it was defined), and \(IRS\) is the corresponding node number in that unit. No chaining is allowed. A shared node must be referenced to the unit where it was first defined.

\(IUS = 0\) implies that the user point is an *auxiliary node*; i.e., it is established without reference to an existing node in a previously-defined unit; \(IRS\) and \(ICS\) are irrelevant in that case.

The next six entries on this record are relevant for auxiliary nodes only. See Figure 7.1 on page 7-6 for details regarding auxiliary nodes.

**XG, YG, ZG**
\((x_g, y_g, z_g)\) global coordinates

**IUVW, IRUVW**
3-digit binary integers indicating freedom or constraint for each of the three translational and each of the three rotational displacement components \((u, v, w)\) and \((ru, rv, rw)\), respectively. The significance of each binary digit is as follows:

\[
\begin{align*}
0 & : \text{fixed} \\
1 & : \text{free}
\end{align*}
\]

The most significant digit corresponds to \(u\) (or \(ru\)), and the least significant corresponds to \(w\) (or \(rw\)). For example, to specify \(u\) fixed with \(v, w\) free, and \(rv\) fixed, with \(ru, rw\) free

\[
\begin{align*}
IUVW & = 011 \\
IRUVW & = 101
\end{align*}
\]

**IUVW** and **IRUVW** are similar to **ITRA** and **IROT** (P-2). See “P-2 Boundary Conditions—Record 2” on page 6-61.

**IAUX**
degree-of-freedom option

\[
\begin{align*}
0 & : \text{the directions of the freedoms are governed by} \\
& (\hat{x}_g, \hat{y}_g, \hat{z}_g) \text{ nodal-global coordinates} \\
1 & : \text{the directions of the freedoms are governed by} \\
& (x_a, y_a, z_a) \text{ nodal-auxiliary coordinates}
\end{align*}
\]

**NPTS**
number of user points specified via this S-1 record; set to unity by **STAGS** if nonpositive or omitted
**ILAY**

layer parameter

0 – ignored (this is the preferred value when the user node is not in a shell unit or when it is in a shell unit that is not tessellated with solid elements)

>0 – the user node is located on surface # ILAY of the specified shell unit (this option should only be used when the shell unit is tessellated with solid elements)

if (NPTS > 1) then  go to  S-1a
else  follow instructions at end of  S-1a
auxiliary node $n$: 

\begin{align*}
(x^n_x, y^n_y, z^n_z) &= (X_G, Y_G, Z_G) \\
\text{point 1, on the } x_a \text{ axis: } q_1 &= (x^1_x, y^1_y, z^1_z) = (Y_A, Y_A, Y_A) \\
\text{point 2, in the } (x'_a, y'_a) \text{ plane: } q_2 &= (x^2_x, y^2_y, z^2_z) = (Y_A, Y_A, Y_A)
\end{align*}

\[ x_a = \frac{q_1}{|q_1|}, \quad z_a = \frac{q_1 \times q_2}{|q_1 \times q_2|}, \quad y_a = z_a \times x_a \]

if \((x'_a, y'_a, z'_a)\) is specified, then \((x'', y'', z'') = (x'_a, y'_a, z'_a)\) else \((x'', y'', z'') = (x^2_x, y^2_y, z^2_z)\)

\((x'', y'', z'')\) computational coordinates \quad \((x^2_x, y^2_y, z^2_z)\) global coordinates

\((x'_a, y'_a, z'_a)\) nodal-auxiliary coordinates \quad \((x^2_x, y^2_y, z^2_z)\) nodal-global coordinates

**Figure 7.1** Degree-of-Freedom Directions at an Auxiliary Node
S-1a User Point Incrementations (upts protocol)

A single record of type S-1a must be included immediately after each type S-1 record on which the NPTS parameter is greater than unity. Seven incrementation variables are specified here for use with the S-1 record looping function.

<table>
<thead>
<tr>
<th>JUPT</th>
<th>JUS</th>
<th>JRS</th>
<th>JCS</th>
<th>Dxg</th>
<th>Dyg</th>
<th>Dzg</th>
</tr>
</thead>
<tbody>
<tr>
<td>JUPT incrementation for the IUPT (user point number) variable, on S-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUS incrementation for the IUS (unit number) variable, on S-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JRS incrementation for the IRS (row/element number) variable, on S-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JCS incrementation for the ICS (column number) variable, on S-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dxg incrementation for the XG (coordinate) variable, on S-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dyg incrementation for the YG (coordinate) variable, on S-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dzg incrementation for the ZG (coordinate) variable, on S-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero, or positive.

**Example:** the following S-1/S-1a record combination:

```
1 0 0 0 0.0 0.0 0.0 111 111 0 3   $ S-1
1 0 0 0 0.2 0.4 -0.2               $ S-1a
```

generates the same three user points as the following three S-1 records:

```
1 0 0 0 0.0 0.0 0.0 111 111 0   $ S-1
2 0 0 0 0.2 0.4 -0.2 111 111 0   $ S-1
3 0 0 0 0.4 0.8 -0.4 111 111 0   $ S-1
```

**IAUX (S-1) degree-of-freedom option**

if ( IAUX > 0 ) then  go to  S-2
else  follow instructions at end of  S-2
**S-2 Auxiliary Coordinate System (upts protocol)**

The S-2 records are used to define auxiliary coordinate systems. These are used for definition of directions of nodal freedoms in element units. Matrices transforming displacements and rotations from \((x_a', y_a', z_a')\) nodal-auxiliary coordinates to \((x_g', y_g', z_g')\) nodal-global coordinates are computed by STAGS and can be utilized in user-written subroutines.

For all nodes in an element unit that do not also belong to a shell unit or to another previously-defined element unit, the displacement and rotation components of the freedoms generated at the node will have the directions indicated by the nodal-auxiliary coordinates if such a system is defined. The purpose is to allow displacement and rotational constraints on components other than those given in the nodal-global coordinates. If an S-2 record is not read, the nodal-global coordinates define the computational coordinates.

The nodal-global coordinate system has its origin at the node point while the coordinate directions coincide with those of the global system. The S-2 record defines the orientation of the auxiliary system by defining two points—\(q_1\), lying on the positive \(x_a\) axis, and \(q_2\), lying anywhere in the \((x_a', y_a')\) plane off the \(x_a\) axis (for example on the \(y_a\) axis). The \(z_a\) axis completes a right-handed Cartesian system. Note that the origins of the two systems, \((x_g', y_g', z_g')\) and \((x_a', y_a', z_a')\), coincide. The algorithm for constructing this system is shown in Figure 7.1 on page 7-6. See Section 4.1 “Coordinate Systems” on page 4-1 for a summary of these and all of the other coordinate systems that STAGS uses.

The user should be very careful about where in the \((x_a', y_a')\) plane \(q_2\) is chosen, since the signs of the auxiliary coordinate directions depend on this selection. The safest strategy is to put \(q_2\) along the \(y_a\) axis itself. Note the similarity of the algorithm for determining the nodal-auxiliary coordinate system here with that for nonlinear mounts (see T-1, for example) and the general algorithm for beams (see T-2, for example). Please refer to Chapter 14 “The Element Library” for more details.

The S-2 record is included only if \(IAUX = 1\) (S-1). If multiple user points are defined via the S-1/S-1a looping function, the default coordinate system or the auxiliary coordinate system defined via this S-2 record must be the same for all of these points.
**XAX, XAY, XAZ** nodal-global coordinates of point \( q_1 \), lying on the \( x_a \) axis. Refer to Figure 7.1 on page 7-6.

**YAX, YAY, YAZ** nodal-global coordinates of point \( q_2 \), lying anywhere in the \((x_a, y_a)\) plane off the \( x_a \) axis; a point on the \( y_a \) axis is recommended.

**NUPT** (H-1) number of user points

**NS5** (H-1) number of contact-line records

**NT1** (H-1) number of “spring” elements

**NT2** (H-1) number of “beam” elements

**NT3** (H-1) number of triangular shell elements

**NT4** (H-1) number of quadrilateral shell elements

**NT5** (H-1) other-elements/element-command flag

if ( NUPT user points have been defined ) then
  if ( NS5 > 0 ) then go to S-5
  elseif ( NT1 > 0 ) then go to T-1
  elseif ( NT2 > 0 ) then go to T-2
  elseif ( NT3 > 0 ) then go to T-3
  elseif ( NT4 > 0 ) then go to T-4
  elseif ( NT5 > 0 ) then go to T-5
  else go to T-100
else return to S-1
S-3 User Points (user–points protocol)

The user–points protocol uses one or more S-3/S-3a/S-4 record sets to establish a nodal point list for the element unit. User-point-definition via this protocol is triggered by setting the NUPT parameter equal to zero or -1 on H-1; and it is terminated by setting the N1 parameter (on an S-3 record) equal to 999999. Additional nodes are defined in user-written subroutine USRPT if IUWP=1 on H-1. If the node point coincides with a node on one of the shell units, its location can be defined through reference to the row and column number for that node. Full displacement compatibility at the node between shell unit and element unit will then be enforced. Other node point locations (auxiliary nodes) are defined by use of their coordinates in the global system.

For nodes that coincide with a node on a shell unit, the directions of the freedoms (displacement and rotation vector components) are identical to those on the shell unit at that node. When more than one shell unit belongs to such a node, the components belong to the shell unit with the lowest number. The same rule applies when two different element units share a node: the components belong to the lowest unit. All shell units have a lower number than any element units. Any node that has not been created before in a lower unit is called an auxiliary node. For auxiliary nodes, the user has a choice regarding the freedom pattern. Often constraints are readily defined if the directions of the freedom coincide with the axes of the global system. For more general cases, auxiliary systems can be defined on an S-4 record.

Constraints on displacements or rotations can be introduced on this record (IUUVW, IRUVW). Attached masses are also permitted. Constraints can also be applied as loads (U-3).

User points may be numbered explicitly by the analyst, or automatically by the program, as explained in the following two paragraphs.

If the very first user point that the analyst defines is given a positive user-point number (N1), then the user-point numbers of all of the user points in the element unit must be specified explicitly. In this case, the user points may be defined in any order—but they must be numbered from 1 to the total number of user points defined. Here, STAGS sets NUPTS equal to the highest user-point number that is specified during this process.

If the first user point that the analyst defines is given a nonpositive N1 value, STAGS ignores that and all subsequent user-point-number specifications, and assigns all user-point numbers sequentially—starting at 1 and incrementing by 1 until the analyst stops the user-point definition process. In this case, STAGS sets NUPT equal to the total number of user points that are defined.
In either event, after all user-point definitions have been made—and all user points identified and/or created by user-written subroutine USRPT have been processed—STAGS checks to ensure that each possible user point IUPT in the range \(1 \leq IUPT \leq NPTS\) has been defined.

If \(IUS > 0\) for any user point, then IRS must be \(> 0\) for that point. If the node belongs to another shell unit \((IUS \leq NUNITS, B-2)\), then ICS must be \(> 0\), giving both the row and column number of the node in the lower unit. If \(IUS > NUNITS\), the current element unit is joined to another element unit with a lower number; in that case IRS is the node number belonging to the lower element unit, defined on a previous S-3 record or in the user-written subroutine USRPT, and ICS = 0. If this is element unit \(l\) counted in the order that it is defined, \(IUS = k\), where \(k = NUNITS + l\), where \(l \leq NUNITE\) (B-2). If \(IUS = 0\), the nodal position must be defined in \((x_g, y_g, z_g)\) global coordinates.

A single S-3 record (with its associated S-4 record, if required) can be used to define each user point. For some configurations, it is convenient to use the S-3 *looping* feature described below to specify two or more closely-related user points simultaneously. The NPTS parameter, on S-3 facilitates that.

In any event, the S-3/S-3a, S-4 sequence may be repeated as many times as necessary to specify user points. This sequence is terminated by setting the IUPT parameter equal to 999999.

<table>
<thead>
<tr>
<th>IUPT</th>
<th>IUS</th>
<th>IRS</th>
<th>ICS</th>
<th>XG</th>
<th>YG</th>
<th>ZG</th>
<th>IUW</th>
<th>IRUVW</th>
<th>IAUX</th>
<th>NPTS</th>
<th>ILAY</th>
</tr>
</thead>
</table>

**IUPT**  
user point (node) number (or end-of-sequence signal):

If \(IUPT < 1\) on the very first S-3 record, it triggers the automatic user-point numbering mode for the upts protocol (in which STAGS sets the number of the first point defined to 1 and increments the number of each subsequent point by 1); the values of IUPT and of its incrementation parameter (JPNT, on S-3a) are ignored on the first and on all subsequent S-3/S-3a record sets.

If \(0 < IUPT < 999999\) on the first S-3 record, it triggers the explicit user-point numbering mode [in which the initial point defined by this S-3 record is user point number IUPT, and the user-point numbers of subsequent points (if any) defined by the first and by subsequent S-3/S-3a records are computed from IUPT and JUPT].

If \(IUPT = 999999\) on any S-3 record, STAGS stops defining user points.

**IUS, IRS, ICS**  
shell unit number, row number, column number for node on a shell unit

If \(IUS > NUNITS, IUS = k = NUNITS + l\), where \(l\) is a previously-defined element unit (counted in the order it was defined), and IRS is the corresponding node.
number in that unit. No chaining is allowed. A shared node must be referenced to
the unit where it was first defined.

**IUS** = 0 implies that the user point is an auxiliary node; i.e., that it is established
in the current element unit without reference to an existing node in a previously-
defined unit; **IRS** and **ICS** are irrelevant in that case.

The next six entries on this record are relevant for auxiliary nodes only.
See Figure 7.1 on page 7-6 for details regarding auxiliary nodes.

**XG, YG, ZG**

Global coordinates

**IUVW, IRUVW**

3-digit binary integers indicating freedom or constraint for each of the three
translational and each of the three rotational displacement components \((u, v, w)\)
and \((ru, rv, rw)\), respectively. The significance of each binary digit is as follows:

- 0—fixed
- 1—free

The most significant digit corresponds to \(u\) (or \(ru\)), and the least significant
corresponds to \(w\) (or \(rw\)). For example, to specify \(u\) fixed with \(v, w\) free, and \(rv\)
fixed, with \(ru, rw\) free

\[
\text{IUVW} = 011 \\
\text{IRUVW} = 101
\]

**IUVW** and **IRUVW** are similar to **ITRA** and **IROT** (P-2).
See “P-2 Boundary Conditions—Record 2” on page 6-61.

**IAUX**

degree-of-freedom option

0 – the directions of the freedoms are governed by

\((x_g, y_g, z_g)\) nodal-global coordinates

1 – the directions of the freedoms are governed by

\((x_u, y_u, z_u)\) nodal-auxiliary coordinates

**NPTS**

Number of user points specified via this S-3 record; set to unity by **STAGS** if
nonpositive or omitted

**ILAY**

Layer parameter

0 – ignored (this is the preferred value when the user node is not in a shell
unit or when it is in a shell unit that is not tessellated with solid elements)

> 0 – the user node is located on surface \(#\) **ILAY** of the specified shell unit
(this option should only be used when the shell unit is tessellated with solid elements)

\[
\text{if ( NPTS > 1 ) then go to S-3a} \\
\text{else follow instructions at end of S-3a}
\]
**S-3a User Point Incrementations (user–points protocol)**

A single record of type S-3a must be included immediately after each type S-3 record on which the NPTS parameter is greater than unity. Seven incrementation variables are specified here for use with the S-3 record looping function.

<table>
<thead>
<tr>
<th>JUPT</th>
<th>JUS</th>
<th>JRS</th>
<th>JCS</th>
<th>Dxg</th>
<th>Dyg</th>
<th>Dzg</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>incrementation for the IUPT (user point number) variable, on S-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>incrementation for the IUS (unit number) variable, on S-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>incrementation for the IRS (row/element number) variable, on S-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>incrementation for the ICS (column number) variable, on S-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>incrementation for the XG (coordinate) variable, on S-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>incrementation for the YG (coordinate) variable, on S-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>incrementation for the ZG (coordinate) variable, on S-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these variables can be negative, zero, or positive.

**Example:** the following S-3/S-3a record combination:

```
1 0 0 0 0.0 0.0 0.0 111 111 0 3 $ S-3
1 0 0 0 0.2 0.4 -0.2 $ S-3a
```

generates the same three user points as the following three S-3 records:

```
1 0 0 0 0.0 0.0 0.0 111 111 0 $ S-3
2 0 0 0 0.2 0.4 -0.2 111 111 0 $ S-3
3 0 0 0 0.4 0.8 -0.4 111 111 0 $ S-3
```

**IAUX (S-3) degree-of-freedom option**

if (IAUX > 0) then go to S-4
else follow instructions at end of S-4
S-4 Auxiliary Coordinate System (user–points protocol)

The S-4 records are used to define auxiliary coordinate systems. These are used for definition of directions of nodal freedoms in element units. Matrices transforming displacements and rotations from \((x_a,y_a,z_a)\) nodal-auxiliary coordinates to \((\check{x}_g,\check{y}_g,\check{z}_g)\) nodal-global coordinates are computed by STAGS and can be utilized in user-written subroutines.

For all nodes in an element unit that do not also belong to a shell unit or to another previously-defined element unit, the displacement and rotation components of the freedoms generated at the node will have the directions indicated by the nodal-auxiliary coordinates if such a system is defined. The purpose is to allow displacement and rotational constraints on components other than those given in the nodal-global coordinates. If an S-4 record is not read, the nodal-global coordinates define the \((x'',y'',z'')\) computational coordinates.

The nodal-global coordinate system has its origin at the node point while the coordinate directions coincide with those of the global system. The S-4 record defines the orientation of the auxiliary system by defining two points—\(q_1\), lying on the positive \(x_a\) axis, and \(q_2\), lying anywhere in the \((x_a,y_a)\) plane off the \(x_a\) axis (for example on the \(y_a\) axis). The \(z_a\) axis completes a right-handed Cartesian system. Note that the origins of the two systems, \((\check{x}_g,\check{y}_g,\check{z}_g)\) and \((x_a,y_a,z_a)\), coincide. The algorithm for constructing this system is shown in Figure 7.1 on page 7-6. See Section 4.1 “Coordinate Systems” on page 4-1 for a summary of these and all of the other coordinate systems that STAGS uses.

The user should be very careful about where in the \((x_a,y_a)\) plane \(q_2\) is chosen, since the signs of the auxiliary coordinate directions depend on this selection. The safest strategy is to put \(q_2\) along the \(y_a\) axis itself. Note the similarity of the algorithm for determining the nodal-auxiliary coordinate system here with that for nonlinear mounts (see T-1, for example) and the general algorithm for beams (see T-2, for example). See Chapter 14 “The Element Library” for more details.

The S-4 record is included only if \(IAUX = 1\) (S-3). If multiple user points are defined via the S-3/S-3a looping function, the default coordinate system or the auxiliary coordinate system defined via this S-4 record must be the same for all of these points.
**XAX, XAY, XAZ**  
nodal-global coordinates of point \(q_1\), lying on the \(x_a\) axis.
Refer to Figure 7.1 on page 7-6.

**YAX, YAY, YAZ**  
nodal-global coordinates of point \(q_2\), lying anywhere in the \((x_a, y_a)\) plane
off the \(x_a\) axis; a point on the \(y_a\) axis is recommended.

---

**IUPT**  
initial user point number  
(S-3)

**NS5**  
number of contact-line records  
(H-1)

**NT1**  
number of “spring” elements  
(H-1)

**NT2**  
number of “beam” elements  
(H-1)

**NT3**  
number of triangular shell elements  
(H-1)

**NT4**  
number of quadrilateral shell elements  
(H-1)

**NT5**  
other-elements/element-command flag  
(H-1)

if \(\text{IUPT} < 999999\) then
return to S-3
elseif \(\text{NS5} > 0\) then  
go to S-5
elseif \(\text{NT1} > 0\) then  
go to T-1
elseif \(\text{NT2} > 0\) then  
go to T-2
elseif \(\text{NT3} > 0\) then  
go to T-3
elseif \(\text{NT4} > 0\) then  
go to T-4
elseif \(\text{NT5} > 0\) then  
go to T-5
else  
go to T-100
7.2 Line-to-Line–Contact Specifications

There are situations that arise during the analysis of lightweight structures where the edges of two shells come into contact with each other, as shown in Figure xx, as a result of deformations. Such contacts will most commonly occur along a crack face in a portion of the structure that undergoes compression and local buckling at loads below the ultimate design strength of the structure. Typically, the opposite sides of the crack displace differently; and opposing edges of the crack contact each other obliquely—with no more than one or a few points of contact along the length of the crack. Line-to-line contact capabilities have been implemented in STAGS to take this phenomenon into account.*

![Contacted Line Contacting Line](image)

**Figure 7.2** Line-to-Line Contact Phenomenon

Lines in a STAGS element unit are also typically connected to node points—some of which may have been defined in previously-specified shell units and/or in one or more previously-specified element units, and some of which may be unique to that element unit. Lines are used in line-contact specifications, as described next. Line and line-contact specifications are required and should only be given if the NS5 parameter is positive on the H-1 record for the current element unit.

* For more information about this, see the “E822 Line contact element” portion of Section 14.8.
S-5 Contact-Line Definition

An S-5 record is used to define a special kind of line on the current element unit (or on a previously-defined shell unit) that may experience line-to-line contact with one or more other lines. This record specifies the contact-line identifier to be used for a line, the unit on which the line is defined, and the number of S-5a or S-5b records that are required for that definition. The potential contact-line interactions for a given STAGS model are specified later, on one or more T-8 and related records (described on page 8-41). It is important for the user to understand that each such interaction involves two contact lines—one of which is considered to be the contacting line and the other of which is considered to be the contacted line. The direction of a contact line is determined by the order in which the nodes defining that line are specified. This is important if the line is ever used as a contacted line, because it uniquely specifies the element(s) that are associated with that line (and which may therefore be contacted by a contacting line).

NS5 (H-1) S-5 records are required.

---

**LINEID** **UNITID** **NRECS**

**LINEID** identifier for the current contact line; each contact-line in the model must have a unique **LINEID** identifier

**UNITID** identifier for the (previously-defined) shell unit on which the contact line is defined, or for the current element unit

**NRECS** number of S-5a records required to define this contact line (if it is on a shell unit), or the number of S-5b records required to define it (if it is on the current element unit)

```plaintext
if ( **UNITID** identifies a shell unit ) then
go to S-5a
else
  go to S-5b
endif
```
**S-5a Contact Line on a Shell Unit**

A *contact line* is a directed line segment that is defined by a sequence of node numbers. When a contact line is on a shell unit, **NRECS** (S-5) records of this type are required to define that line. A contact line on a shell unit can be all or part of an edge, all or part of any row or column, or any set of nodes that are connected *via* an unbroken sequence of node numbers. Contact lines are used by **STAGS** in **E822** line-contact-interaction definitions (T-8), each of which identifies a contact that may occur between two free edges. One of these edges is represented by a *contacted* line, which has elements associated with it; and the other is represented as a *contacting* line, which may or may not have elements associated with it. The theory behind line-to-line contact is given by Rankin *et al.* and is summarized in Chapter 14.

The order in which the nodes defining a contact line are specified determines its *direction*. This is important because **STAGS** associates with the contact line the elements that are attached to its left side. Any contact line that is ever used as a *contacted* line *must* have elements associated with it. Any contact line that is only used as a *contacting* line may but need not have elements associated with it. Figure 7.3 shows a typical shell unit in terms of its row and column nodal topology, the shell elements that are defined on that unit, four contact lines, and the elements that are associated with each contact line. Note that line 4 (a *contacting* line) has no elements associated with it.

![Figure 7.3 Contact Lines on a Typical Shell Unit](image)

II JJ NPTS INC

II, JJ  (row,col) indices for a node on the shell unit; if II = 0, all rows in column JJ are included (STAGS resetting NPTS to NROWS, the number of rows in the shell unit); if JJ = 0, all columns in row II are included (STAGS resetting NPTS to NCOLS, the number of columns in the shell unit); II and JJ cannot both be zero; each of the row indices specified by II and its offspring (via NPTS and INC, as described below) must be in the range \(1 \leq II \leq \text{NROWS}\); and each of the column indices specified by JJ and its offspring (via NPTS and INC) must be in the range \(1 \leq JJ \leq \text{NCOLS}\).

NPTS number of nodes to be specified with information on this S-5a record

INC direction indicator, used when II or JJ is 0 or when NPTS > 0:

\[
\begin{align*}
= +1 & \quad \text{increment the row index II by } +1 \\
= -1 & \quad \text{increment the row index II by } -1 \\
= +2 & \quad \text{increment the col index JJ by } +1 \\
= -2 & \quad \text{increment the col index JJ by } -1
\end{align*}
\]

Example 1: The following S-5a record defines contact line 2 on Figure 7.3:

```
2 3 4 1 $ S-5a
```

Example 2: The following S-5a record defines contact line 1 on Figure 7.3:

```
5 5 5 -1 $ S-5a
```

Example 3: The following pair of S-5a records defines defines contact line 3 on Figure 7.3, which starts at node (3,1), proceeds along column 1 to node (7,1) of the shell unit, then makes a left-hand turn and goes along row 7 to node (7,4):

```
3 1 5 1 $ S-5a # 1
7 2 3 2 $ S-5a # 2
```

NS5 \( (H-1) \) number of S-5 records for the current element unit

NRECS \( (S-5) \) number of S-5a records required to define the current line

if \( \) (fewer than NRECS S-5a records have been processed) then

process another S-5a record

elseif \( \) (fewer than NS5 S-5 records have been processed) then

return to S-5

else

follow instructions at end of S-5b
endif
**S-5b Contact Line on an Element Unit**

A *contact line* is a directed line segment that is defined by a sequence of node numbers. When a contact line is on an element unit, **NRECS (S-5)** records of this type are required to specify the user nodes that define that line.

As noted in the S-5a description, above, contact lines are used by **STAGS** in **E822** line-contact-interaction definitions (T-8), each of which identifies a possible contact of two free edges. One of the two edges is treated as the *contacted* line, which must have elements associated with it; and the other is treated as the *contacting* line, which may or may not have elements associated with it.

The order in which the nodes defining a contact line are specified determines the *direction* of that contact line—the tail of the line being its first node and the head being its last. In its line-contact operations, **STAGS** uses the elements that are attached to the *left* side of the *contacted* line, as shown in the following figure:

![Figure 7.4](image.png)  

**Figure 7.4** Contact Line on a Typical Element Unit

<table>
<thead>
<tr>
<th>II NPTS INC</th>
</tr>
</thead>
<tbody>
<tr>
<td>II</td>
</tr>
<tr>
<td>user node number for a node on the contact line</td>
</tr>
</tbody>
</table>
**NPTS** number of nodes to be specified with information on this S-5a record

**INC** incrementation, which may be positive or negative

**NRECS** (S-5) number of S-5a records required to define the current line

**NS5** (H-1) number of S-5 records for the current element unit

**NT1** (H-1) number of “spring” elements

**NT2** (H-1) number of “beam” elements

**NT3** (H-1) number of triangular shell elements

**NT4** (H-1) number of quadrilateral shell elements

**NT5** (H-1) other-elements/element-command flag

if ( fewer than **NRECS** S-5b records have been processed ) then
  process another S-5b record
elseif ( fewer than **NS5** S-5 records have been processed ) then
  return to S-5
else
  if ( **NT1** > 0 ) then go to T-1
  elseif ( **NT2** > 0 ) then go to T-2
  elseif ( **NT3** > 0 ) then go to T-3
  elseif ( **NT4** > 0 ) then go to T-4
  elseif ( **NT5** > 0 ) then go to T-5
  else go to T-100
endif
This chapter describes the historic Edef protocol for specifying some or all of the elements that are to be used in constructing an element unit in a STAGS model. As noted earlier, this protocol is used if and only if one or more of the NT1, NT2, NT3, NT4 and NT5 parameters is positive on the H-1 record for the current element unit. If all of these five parameters are nonpositive, then STAGS expects the analyst to specify all of the elements for the element unit via the Ecom protocol, which is described in Chapter 9.

The Edef protocol is a rigorous procedure that begins by using the values of NT1, NT2, NT3 and NT4 (on H-1) to control loops in which appropriate type T-x record sets are processed to make NT1 spring element definitions, then to make NT2 beam element definitions, then to make NT3 triangular element definitions, and finally to make NT4 quadrilateral element definitions. A zero value for any (or all) of these parameters indicates that no elements of that type are to be defined (via “regular” input records). After doing that, STAGS examines the NT5 parameter (on H-1) to determine whether or not additional type T-x and/or T-xx record sets are then required to define one or more contact, sandwich and/or solid elements. STAGS also checks to determine whether or not the Ecom protocol is to be used after that, to define one or more elements of any type.

8.1 Definition of “Spring” Elements via the Edef Protocol

If and only if NT1 is positive on H-1, STAGS expects the analyst to specify NT1 “spring” elements at this point in his or her model-definition input file and will attempt to read and process one or more type T-1 record sets to define those elements. Each of these T-1 record sets defines one or more type E110 (mount), E120 (rigid-link), E121 (soft-link) or E130 (generalized-fastener) spring elements—according to the value of the KELT parameter on T-1.
**T-1 Spring Element**

As noted above, STAGS expects $NT_1$ “spring” element definitions at this point in the model-definition input file if $NT_1 > 0$ on the H-1 record for this element unit. The $NT_1$ parameter specifies the total number of elements in the four categories that are listed below. Typically, each spring is defined individually on a pair of records, as follows:

- **E110 mount** $T-1 / T-1a$
- **E120 rigid link** $T-1 / T-1b$
- **E121 soft link** $T-1 / T-1b$
- **E130 generalized fastener** $T-1 / T-1c$

Under some circumstances, the user will find it convenient to employ the *looping* feature provided on record T-1 via the $NX$ parameter, with increments for one or more of the nodes specified thereon.

In any event, the T-1 series must be repeated as many times as necessary to specify a total of $NT_1$ “spring” elements. Please refer to “E110 Mount element” on page 14-6, to “E120 Rigid link element” on page 14-7, to “E121 Soft link element” on page 14-7 and/or to Section 14.8 “Contact Elements” on page 14-51 as necessary for explanations of the input data described here.

In the rigid link element (element type **E120**), the third node $N_3$ is a reference node that is used to enforce rotational compatibility between $N_1$ and $N_2$. If $N_3$ is zero, the freedoms on $N_2$ are independent of those on $N_1$—a condition that simulates a ball joint at $N_2$. If $N_3$ is nonzero, total rotational compatibility is enforced between $N_1$ and $N_2$—a condition that simulates the ordinary rigid link. If $N_3$ is positive, the rotational constraint is enforced with a set of three Lagrange multipliers. For this option, computational coordinates for each node in the link are completely independent (at the discretion of the user), affording maximum flexibility. If $N_3$ is negative, the rotational constraint is enforced using partial compatibility (see record G-2, with $ID_1 = -2$). A restriction is introduced that forces the computational coordinates of $N_2$ to be the same as for the master node to which $N_1$ refers. This can have unforeseen consequences if rigid links cause new master/slave relationships to be defined between other nodes in the model. It is recommended that a negative value for $N_3$ be used with caution.

In the soft link element (element type **E121**), node $N_1$ is usually associated with a node in a shell unit; and $N_2$ and $N_3$ are usually associated with different layers of a solid.
<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>KELT</th>
<th>NX</th>
<th>INC1</th>
<th>INC2</th>
<th>INC3</th>
<th>USERELT</th>
<th>INC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>node 1</td>
<td>node 2</td>
<td>node 3:</td>
<td>this is a <em>reference node</em> for the <strong>E120</strong> element (a positive value enforces rotational compatibility between nodes N1 and N2);</td>
<td>this is an <em>active soft-link node</em> for the <strong>E121</strong> element</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KELT</td>
<td>110 – <strong>E110</strong> mount element</td>
<td>120 – <strong>E120</strong> rigid link element</td>
<td>121 – <strong>E121</strong> soft link element</td>
<td>130 – <strong>E130</strong> generalized fastener</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NX</td>
<td>number of “spring” elements specified <em>via</em> the current T-1/* series;</td>
<td>set equal to unity by STAGS if nonpositive or omitted</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC1</td>
<td>incrementation variable for N1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC2</td>
<td>incrementation variable for N2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC3</td>
<td>incrementation variable for N3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USERELT</td>
<td>user-specified element number (only used if IUWLE=1 on H-1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC4</td>
<td>incrementation variable for USERELT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
if ( KELT = 110 ) then go to T-1a
elseif ( KELT = 120 ) then go to T-1b
elseif ( KELT = 121 ) then go to T-1b
elseif ( KELT = 130 ) then go to T-1c
```
**T-1a Mount**

If the *looping* feature on the T-1 record is *not* used, each E110 mount element is defined individually by a T-1 record followed by a T-1a record. If the *looping* feature *is* used, a series of mount elements is defined by the T-1 record followed by **nx** T-1a records—one for each element of the series.

The mount element is a special nonlinear spring capable of modeling a user-defined displacement-velocity-force profile. The mount is discussed in “I-4a Mount Element Table Size” on page 5-64 and in “E110 Mount element” on page 14-6. Please refer to those sections as necessary. As can be seen in Figure 14.1 on page 14-6, rigid links can be introduced. Rigid links provide a method for defining rotational stiffness in addition to axial spring stiffness. Rigid links are expressed in local element coordinate systems, \((X_1, Y_1, Z_1)\) and \((X_2, Y_2, Z_2)\), as shown in Figure 14.1. The algorithm for determining these systems is very similar to that discussed for record S-2, with the reference node N3 playing the same role as the point YAX, YAY, YAZ (S-2). Clearly, node N3 must not lie on the same line as N1 and N2. The coordinate \(X_1\) is in the direction of the line connecting N1 to N2. The \(Y_1\) axis is normal to \(X_1\) and in the plane defined by N1, N2, and N3, pointing “toward” N3. Finally, \(Z_1\) is normal to the plane containing N1, N2, and N3, completing a right-handed system. N3 can be either a structural node or a dummy node, defined only for reference to the mount. Any freedoms defined on dummy nodes are ignored. For rigid link 1, the origin of the \((X_1, Y_1, Z_1)\) system is situated at N1, and the distances RLX1, RLY1, and RLZ1 are expressed in this system. For rigid link 2, the \((X_2, Y_2, Z_2)\) system, with origin at N2, is used to express the distances RLX2, RLY2, RLZ2. These distances can be positive, negative, or zero. When all these distances are zero, no rigid links exist.
<table>
<thead>
<tr>
<th>IMNT1</th>
<th>IMNT2</th>
<th>RLX1</th>
<th>RLY1</th>
<th>RLZ1</th>
<th>RLX2</th>
<th>RLY2</th>
<th>RLZ2</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMNT1</td>
<td>Mount Element Table identifier (I-4a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMNT2</td>
<td>optional additional mount table identifier: if IMNT2 is not zero, then IMNT1 and IMNT2 must refer to tables consisting of a single row (displacement table, NRV = 1, I-4a) or column (velocity table, NRD = 1, I-4a). There must be one table of each type, or an error will result. If IMNT2 &gt; 0, the total mount force is the sum of the force from the displacement table and the force from the velocity table.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLX1, RLY1, RLZ1</td>
<td>((X_1, Y_1, Z_1)) coordinates of point RL1; refer to Figure 14.1 on page 14-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLX2, RLY2, RLZ2</td>
<td>((X_2, Y_2, Z_2)) coordinates of point RL2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*follow instructions at end of T-1c*
**T-1b Rigid or Soft Link**

If the *looping* feature on the T-120a record is *not* used, each **E120** rigid link element or **E121** soft link element is defined individually by a T-1 record followed by a T-1b record. If the *looping* feature *is* used, a series of rigid link or soft link elements is defined by the T-1 record, followed by one T-1b record on which the **SCALE** variable to be used for *all* of the elements of the series is specified.

The rigid link element is discussed in “E120 Rigid link element” on page 14-7. A rigid link element constrains the distance between two nodes (**N1** and **N2**) to be invariant during an analysis. The displacements of node **N2** are dependent on the displacements and rotations of node **N1** through a rigid-link constraint equation, which is enforced *via* Lagrange multipliers.

The potential energy $\Pi$ for a rigid link element is given by

$$\Pi = \phi \alpha^T (x_1 + r - x_2)$$

where $\alpha$ is a vector of three Lagrange multipliers, $x_1$ and $x_2$ are the current global positions of nodes **N1** and **N2**, respectively, and $r$ is the directed position of **N2** with respect to **N1**. The magnitude of $r$ is the original distance between nodes **N1** and **N2**. The user-specified element scale factor $\phi$ is employed to make the magnitude of the stiffness contributions of the rigid link element comparable in size to those of other elements in the configuration—this value is typically on the order of the elastic modulus of the material.

Note that the rotations on **N2** are unaffected unless the reference node **N3** is nonzero, in which case G-2 records are automatically generated to constrain the rotations on **N1** to be the same as **N2**; separate G-2 records can always be generated to constrain the rotational freedoms as the user may desire.

The soft link element is discussed in “E121 Soft link element” on page 14-7. A soft link element constrains the three nodes associated with it to be colinear.

---

**SCALE**

**SCALE** element scale factor

*follow instructions at end of T-1c*
**T-1c Generalized Fastener**

If the *looping* feature on the T-1 record is *not* used, each **E130** generalized fastener element is defined individually by a T-1 record followed by a T-1c record. If the *looping* feature *is* used, a series of generalized fastener elements is defined by the T-1 record followed by a *single* T-1c record on which the mount-table-identifier and material-code variables to be used for *all* of the elements of the series are specified.

The generalized fastener element is closely related to the **E110** (mount) element, defined on T-1/ T-1a. Please refer to “E121 Soft link element” on page 14-7 for element-formulation details and to Figure 8.1 “Degree-of-Freedom Directions at an Auxiliary Node” on page 8-12.

<table>
<thead>
<tr>
<th>IMNT1</th>
<th>IMNT2</th>
<th>IMNT3</th>
<th>IMNT4</th>
<th>IMNT5</th>
<th>IMNT6</th>
<th>PLAS1</th>
<th>PLAS2</th>
<th>PLAS3</th>
<th>PLAS4</th>
<th>PLAS5</th>
<th>PLAS6</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMNT1</td>
<td>Mount Table identifier (see l–4a) applied to the relative local x translation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMNT2</td>
<td>Mount Table identifier (see l–4a) applied to the relative local y translation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMNT3</td>
<td>Mount Table identifier (see l–4a) applied to the relative local z translation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMNT4</td>
<td>Mount Table identifier (see l–4a) applied to the relative local ( \theta_x ) rotation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMNT5</td>
<td>Mount Table identifier (see l–4a) applied to the relative local ( \theta_y ) rotation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMNT6</td>
<td>Mount Table identifier (see l–4a) applied to the relative local ( \theta_z ) rotation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLAS1</td>
<td>breakage code for <strong>IMNT1</strong>: see discussion, below, for significance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLAS2</td>
<td>breakage code for <strong>IMNT2</strong>: see discussion, below, for significance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLAS3</td>
<td>breakage code for <strong>IMNT3</strong>: see discussion, below, for significance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLAS4</td>
<td>breakage code for <strong>IMNT4</strong>: see discussion, below, for significance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLAS5</td>
<td>breakage code for <strong>IMNT5</strong>: see discussion, below, for significance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLAS6</td>
<td>breakage code for <strong>IMNT6</strong>: see discussion, below, for significance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**STAGS** uses the six breakage codes, **PLASi**, specified here to control generalized fastener element breakage modes for the associated mount identifiers, **IMNTi** (where \( 1 \leq i \leq 6 \)). Each breakage code may have one of the following five values:

- **PLASj** = 1 \( \Rightarrow \) fastener \( j \) behaves elastically; its failure *does not* cause other fasteners to fail
- **PLASj** = 2 \( \Rightarrow \) fastener \( j \) behaves elastically; its failure causes *all* of the fasteners to fail
- **PLASj** = 3 \( \Rightarrow \) fastener \( j \) behaves elastically, and is unbreakable, throughout the analysis
- **PLASj** = 4 \( \Rightarrow \) fastener \( j \) behaves plastically; its failure *does not* cause other fasteners to fail
- **PLASj** = 5 \( \Rightarrow \) fastener \( j \) behaves plastically; its failure causes *all* of the fasteners to fail
Model Input—Element Units (2)  

Definition of “Spring” Elements via the Edef Protocol

- **NT1** (H-1) number of “spring” elements
- **NT2** (H-1) number of “beam” elements
- **NT3** (H-1) number of triangular shell elements
- **NT4** (H-1) number of quadrilateral shell elements
- **NT5** (H-1) other-elements/element-command flag

if ( fewer than **NT1** spring elements have been defined ) return to **T-1**

elseif ( **NT2** > 0 ) then go to **T-2**

elseif ( **NT3** > 0 ) then go to **T-3**

elseif ( **NT4** > 0 ) then go to **T-4**

elseif ( **NT5** = 1 ) then go to **T-5**

elseif ( **NT5** = 2 ) then go to **T-5**

elseif ( **NT5** = 3 ) then go to **T-100**

else go to **U-1**
8.2 Definition of “Beam” Elements via the Edef Protocol

If and only if NT2 is positive on H-1, STAGS expects the analyst to specify NT2 “beam” elements at this point in his or her model-definition input file and will attempt to read and process one or more type T-2 record sets to define those elements. Each of these T-2 record sets defines one or more type E210 (beam), or E250 (planar boundary conditions) “elements”—according to the value of the KELT parameter on T-2.

T-2 Beam

If and only if NT2 > 0 on the H-1 record for this element unit, STAGS expects NT2 “beam” element definitions at this point in the model-definition input file. In this case, NT2 specifies the total number of elements in the two categories that are listed below. Typically, each “beam” element is defined individually on one or two records, as follows:

- E220 standard beam element T-2 / T-2a
- E250 planar BC element T-2

Under some circumstances, the user will find it convenient to employ the looping feature provided on record T-2 via the NX parameter, with increments specified thereon for one or more of the listed nodes. In any event, the T-2 series of records must be repeated as many times as necessary to specify a total of NT2 “beam” elements.

Beam References

- Section 4.1 “Coordinate Systems” on page 4-1
- Section 14.4 ““Beam” Elements” on page 14-12
- Figure 8.5 on page 8-33

As shown in Figure 14.5 on page 14-13, an (x′, y′, z′) element coordinate system, with its origin located at node 1, is used to define beams. A reference node is required to establish the element coordinate system. The y′ coordinate is normal to x′ and in the plane defined by N1, N2, N3, pointing “toward” N3. Finally, z′ is normal to the plane containing N1, N2, N3 and completes a right-handed system. N3 can be either a structural node or a dummy node, defined only for reference to the beam. All freedoms at a dummy node should be suppressed on the S-1 record. Note the similarity of the (x′, y′, z′) system with the system for nonlinear mounts (T-1/T-1a) and the (x′a, y′a, z′a) nodal-auxiliary coordinate system defined on the S-2 record.
Moving-Plane Boundary References

- Section 14.8 “Contact Elements” on page 14-51
- “B-2 General Model Summary” on page 5-12
- “O-1a Discrete Ring—Record 1” on page 6-50

The T-2 record can also be used to define a moving plane boundary which is like a symmetry boundary, except that the symmetry plane itself is allowed to move as a rigid body. In the element unit, beam-like elements are strung along a user-defined space curve forming the boundary. It is the user’s responsibility to make sure that curve initially lies in a plane (if the constraint is violated initially, STAGS will complain). The reference node N3 must also lie in the boundary plane.

Typically, each beam element within an element unit will be defined individually with its own T-2 record, requiring the T-2 record to be repeated NT2 times (H-1). Under some circumstances, the user will find it convenient to employ the looping feature triggered by the NX parameter on the T-2 record: in this case, incrementation variables must be specified on a T-2a record following any T-2 record that defines more than one beam; and T-2 records (plus T-2a records, where appropriate) sufficient to define NT2 beam elements must be supplied.

---

N1 N2 N3 KELT ICROSS XSI ECY ECZ ILIN IPLAS NX USERELT

| N1 | node 1; refer to Figure 14.5 on page 14-13 |
| N2 | node 2 |
| N3 | node 3, the reference node, which is used to generate the \((x', y', z')\) element coordinate system; for the moving plane boundary, N3 must lie in the boundary plane. |
| KELT | beam element code number; see Section 14.4 “‘Beam’ Elements” on page 14-12. The only “beam” elements currently available are: E210 Beam and E250 Planar Boundary Condition. |
| ICROSS | cross-section number, as defined by ITAB (J-1), in the Cross Section Table: ICROSS = 0 indicates that the stiffener is defined in user-written CROSS. ICROSS = -1 indicates that the stiffener line is a moving plane boundary. When ICROSS = 0, data tagged by ✤ below are defined in CROSS; they are automatically initialized to zero before CROSS is called, thereby superseding any nonzero values input here; when ICROSS = -1, only ECY is relevant. Refer to Figure 8.5 on page 8-33 for an illustration of XSI, ECY, and ECZ; compare with Figure 6.5 on page 6-52, the shell-unit-stiffener counterpart. |
Definition of “Beam” Elements via the Edef Protocol

Model Input—Element Units (2)

- **XSI**
  angle $\xi$, in degrees, between the element normal $z'$ and the cross section $z$. $\xi$ is a right-handed rotation about $x$, the longitudinal axis of the beam, which is parallel to $x'$.

- **ECY**
  eccentricity in the $y'$ direction; $ECY$ is the $y'$ coordinate of the $(y', z')$ pair which positions the origin of the $(y, z)$ system.
  For $ICROSS = -1$ (moving plane boundary), $ECY$ is a scale factor used for the numerical conditioning of Lagrange constraints introduced by the multipoint constraint. In that case, $ECY$ should be of the same order of magnitude as entries in the stiffness matrix. A good guess is the modulus of elasticity of the material.

- **ECZ**
  eccentricity in the $z'$ direction; $ECZ$ is the $z'$ coordinate of the $(y', z')$ pair which positions the origin of the $(y, z)$ system.

- **ILIN**
  geometric nonlinearity flag
  
  0 – nonlinear strain-displacement relations
  1 – linear strain-displacement relations

- **IPLAS**
  material nonlinearity flag (see M-5)
  
  0 – linear elastic constitutive relations
  1 – plasticity included
  2 – centroidal plasticity

- **NX**
  number of beam elements to be generated via this T-2 record;
  set to unity by STAGS if nonpositive or omitted

- **USERELT**
  user-specified element number, used only if $IUWLE = 1$ on H-1

- if ($NX > 1$) then go to T-2a
  else follow instructions at end of T-2a
Figure 8.1  Degree-of-Freedom Directions at an Auxiliary Node

Compare with Figure 5.5 “Beam cross sections.” on page 5-122. Also, see 16.2 “Beam Results” on page 16-8.
**T-2a Beam Incrementations**

A single record of type T-2a must be included immediately after each type T-2 record on which the NX parameter is greater than unity. Incrementation variables are specified here for use with the T-2 record looping functions.

<table>
<thead>
<tr>
<th>INC1</th>
<th>INC2</th>
<th>INC3</th>
<th>INC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>INC1</td>
<td>incrementation variable for use with the N1 (node 1) variable on T-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC2</td>
<td>incrementation variable for use with N2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC3</td>
<td>incrementation variable for use with N3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC4</td>
<td>incrementation variable for use with USERELT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero or positive.

**Example:** the following T-2/T-2a record combination:

```
 20 51 99 210 3 0.0 0.0 0.0 0 0 3   $ T-2  record
 5 10 0                           $ T-2a record
```

generates the same three beams as the following three individual T-2 records:

```
 20 51 99 210 3 0.0 0.0 0.0 0 0     $ T-2  record
 25 61 99 210 3 0.0 0.0 0.0 0 0     $ T-2  record
 30 71 99 210 3 0.0 0.0 0.0 0 0     $ T-2  record
```

- **NT2** (H-1) number of “beam” elements
- **NT3** (H-1) number of triangular shell elements
- **NT4** (H-1) number of quadrilateral shell elements
- **NT5** (H-1) other-elements/element-command flag

if ( **NT2** beams have been defined ) then
  if ( **NT3** > 0 ) then go to T-3
  elseif ( **NT4** > 0 ) then go to T-4
  elseif ( **NT5** = 1 ) then go to T-5
  elseif ( **NT5** = 2 ) then go to T-5
  elseif ( **NT5** = 3 ) then go to T-100
  else go to U-1
else continue defining T-2
8.3 Definition of Triangle Elements via the Edef Protocol

If and only if $NT_3$ is positive on H-1, STAGS expects the analyst to specify $NT_3$ triangle elements at this point in his or her model-definition input file and will attempt to read and process one or more type T-3 record sets to define those elements. Each of these T-3 record sets defines one or more $E_{320}$ or $E_{330}$ triangle elements—according to the value of the $KELT$ parameter on T-3.

**T-3 Triangular Shell**

If and only if $NT_3 > 0$ on the H-1 record for this element unit, STAGS expects $NT_3$ triangle element definitions at this point in the model-definition input file. In this case, $NT_3$ specifies the total number of elements in the two categories that are listed below. Typically, each triangle is defined individually by a single T-3 record—or by a T-3/T-3a pair of records, if the IANG wall-reference option is exercised—as follows:

- $E_{320}$ standard triangle
- $E_{330}$ MIN3 triangle

Refer to the discussion in “T-4 Quadrilateral Shell” on page 8-17.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>KELT</th>
<th>I WALL</th>
<th>Z ETA</th>
<th>ECZ</th>
<th>ILIN</th>
<th>IPLAS</th>
<th>IANG</th>
<th>USERELT</th>
</tr>
</thead>
</table>
| N1 | node 1; see Section 14.5 “Shell and Mesh-Transition Shell Elements” on page 14-19
| N2 | node 2
| N3 | node 3
| KELT | element code number (see Chapter 14 “The Element Library”):
|     | = 320 — type $E_{320}$ triangular element
|     | = 330 — type $E_{330}$ triangular element

8-14 | April, 2009 | STAGS 5.0 User Manual
IWALL  wall fabrication identifier:
  0  – shell wall properties are given in user-written subroutine WALL
  >0  – shell wall configuration (fabrication) number in the
        Wall Fabrication Table (K-1)
  <0  – shell fabrication identifier in the GCP Fabrications Table (I-21a):
        *this option is only available for type E330 elements*
When IWALL = 0, data below indicated by ✤ are defined in subroutine
WALL; they are automatically initialized to zero before WALL is called,
thereby superseding any nonzero values input here.

✤ ZETA  angle $\zeta$ between the wall-reference coordinate $x_w$ and the fabrication
coordinate $x$; $\zeta$ is a right-handed rotation about $z$.
See Figure 8.2 on page 8-19.

✤ ECZ  eccentricity in $z'$ direction. ECZ is the $z'$ coordinate of the shell wall middle
surface; refer to Figure 6.2 on page 6-28. In element units, ($x', y', z'$) are used in
place of ($X', Y', Z'$), which do not exist in an element unit. See “Effects of
Eccentricity” on page 16-6.

✤ ILIN  governs geometric nonlinearity
  0  – nonlinear strain-displacement relations
  1  – linear strain-displacement relations
Note that with ILIN = 1, bifurcation buckling is suppressed in the shell unit.

✤ IPLAS  governs material nonlinearity
  0  – elastic behavior only
  1  – plasticity included, with the material law satisfied
       at each element integration point
  2  – plasticity included, with the material law satisfied
       at the element centroid (centroidal plasticity)

IANG  wall-reference option; see discussion above
  0  – use default strategy of projecting $x_p, y_g$ to establish $x_w$
  1  – input $r_w$, which is projected to establish $x_w$

USERELT  user-specified element number, used only when IUWLE = 1 on H-1

if ( IANG = 1 ) then  go to  T-3a
else  follow instructions at end of  T-3a
**T-3a Wall Reference Vector**

RX, RY, RZ

RX, RY, RZ wall reference vector, $\mathbf{r}_w$, which is projected onto the element surface to determine the direction of the wall-reference coordinate $x_w$; $\mathbf{r}_w$ is expressed in $(x_g, y_g, z_g)$ global coordinates.

- NT3 (H-1) number of triangular shell elements
- NT4 (H-1) number of quadrilateral shell elements
- NT5 (H-1) other-elements/element-command flag

if ( fewer than NT3 triangles have been defined ) return to T-3
elseif ( NT4 > 0 ) then go to T-4
elseif ( NT5 = 1 ) then go to T-5
elseif ( NT5 = 2 ) then go to T-5
elseif ( NT5 = 3 ) then go to T-100
else go to U-1
8.4 Definition of “Quadrilateral” Elements via the Edef Protocol

If and only if \( \text{NT4} \) is positive on \( \text{H-1} \), STAGS expects the analyst to specify \( \text{NT4} \) “quadrilateral” elements next and will attempt to read and process one or more type T-4 record sets to define those elements. Each of these T-4 record sets defines one or more type E410, E411 or E480 quad elements, one or more E510 or E710 quad transition elements, or one or more E420 or E430 pairs of E320 or E330 triangular elements—according to the value of the \text{KELT} parameter on T-4.

**T-4 Quadrilateral Shell**

If and only if \( \text{NT4} > 0 \) on the H-1 record for this element unit, STAGS expects \( \text{NT4} \) “quadrilateral” element specifications at this point in the model-definition input file. In this case, \( \text{NT4} \) specifies the total number of elements in the three categories that are listed below. Each quadrilateral is defined individually via a single T-4 record or (for the E480 9-node quadrilateral element) via a T-4/T-4a pair of records—followed by a T-4b record if the \text{IANG} wall-reference option is exercised—as follows:

- E410 4-node quadrilateral element T-4 or T-4 / T-4b
- E411 4-node quadrilateral element T-4 or T-4 / T-4b
- E480 9-node quadrilateral element T-4 / T-4a or T-4 / T-4a / T-4b

\( \text{NT4} \) (H-1) is the total number of quadrilateral shell elements in the current element unit.

**Quadrilateral Shell References**

- Section 4.1 “Coordinate Systems” on page 4-1
- Section 14.2 “Algorithm for Determining the Element Frame” on page 14-2
- Section 14.5 “Shell and Mesh-Transition Shell Elements” on page 14-19
- “K-1 Shell Wall Properties” on page 5-129
- “M-5 Shell Wall” on page 6-24

A quadrilateral shell element is defined by specifying the four corner nodes in counterclockwise order as viewed from above. In this context, “viewed from above” means looking down onto the top surface of the element. As shown in Figure 6.2 on page 6-28, the top surface corresponds to \( Z' = Z'_{\text{max}} \) \((z' = z'_{\text{max}}\) in an element unit). These four nodes, input on T-4, determine the \((x', y', z')\) element coordinate system. Refer to 14.2 and 14.5 for details.
For elements having more than four nodes, nodes 5—\(n\), where \(n\) is the total number of nodes in the element, are specified on T-4a. The order in which the nodes are given is according to the numbering shown in the sketches in Section 14.5 “Shell and Mesh-Transition Shell Elements”. Some 4-node elements, like \(E411\), contain midside deviational nodes that are added automatically, without user input. Currently, only the 9-node \(E480\) element requires a T-4a record.

The \(z'\) axis defines the direction in which pressure acts—a positive pressure value acts in the positive \(z'\) direction, and a negative pressure value acts in the negative \(z'\) direction.

In shell units, the wall-fabrication orientation is determined by \texttt{ZETA} \& \texttt{ECZ} (M-5). Figure 6.2 shows how \texttt{ECZ} defines the eccentricity and how \texttt{ZETA} rotates the \((\bar{x}, \bar{y})\) fabrication coordinates to establish their directions relative to the \((x_w, y_w)\) wall-reference coordinates. Implicit in Figure 6.2 is the shell-unit convention that the \((x_w, y_w)\) wall-reference coordinates are defined to be coincident with the \((x', y')\) shell coordinates. In element units, shell coordinate systems do not exist. Instead, the user is given two options for establishing wall-reference coordinates.

In the default option, \texttt{STAGS} determines whether the \(x_g\) or \(y_g\) axis lies closer to the element plane. If \(x_g\) lies closer, it is projected onto the element surface to establish the \(x_w\) axis. If \(y_g\) lies closer, it is projected to establish the \(y_w\) axis. This means that when \(x_g\) and \(y_g\) both lie in the plane of the element, the result will be the same regardless of which is chosen to be projected.

The other option is to input a wall reference vector, \(r_w\), that determines the direction of \(x_w\). Although the user will in most cases try to make sure that this vector is tangent to the surface he has in mind, the code makes no such assumption. \texttt{STAGS} will project this vector onto the element surface to establish the \(x_w\) axis.

After the \(x_w\) axis is established, using either of the two options, the wall orientation is then determined just as it is for shell units. Compare Figure 8.2 with Figure 6.2. \texttt{ZETA} has the same meaning in both instances. The only difference is that for shell units, the \((x_w, y_w)\) wall-reference coordinates are uniquely defined by the \((x', y')\) shell coordinates, and for element units, one of two user-selected options (\texttt{IANG}, on the T-4, T-9, T-10, T-11, T-12 and T-13 records) is used to establish the wall-reference system.

The process outlined here often allows the user to specify all the wall angles with very simple input. For example, some standard geometries such as cylinders or cones have an axis of revolution (the generator) whose projection onto the element surface lies along the same direction, independent of the element location. If user input specifies the generator as \(r_w\), or if the default is used (for cone angles less than 45°), a unique angle \texttt{ZETA} gives the proper offset.
Figure 8.2  Shell Wall Orientation for Element Units

for all elements in the geometry. Figure 8.3 on page 8-20 shows an example where the generator of a cone is specified as $r_w$. It is usually straightforward to specify the global coordinates of a vector parallel to a shell-of-revolution generator. The user should be careful for more complex geometries. It should also be noted that the user need not know the $(x', y', z')$ shell element coordinate system.

For all quadrilaterals, regardless of the number of nodes or the order of the element, the plane of the element is determined by the procedure outlined in Section 14.2 “Algorithm for Determining the Element Frame”. The result of this procedure is an element reference plane that is the “best fit” to the four corner nodes. $(x_w, y_w)$ wall-reference coordinates, and hence $(x, y)$ fabrication coordinates and $(\phi_1, \phi_2)$ material coordinates, all lie in this “average” plane.

Note that the fabrication coordinate system is oriented by rotating the wall-reference system through the angle $\zeta$. The material coordinate system for each layer in a laminate is then determined by rotating the fabrication system through a unique angle $\zeta_l$ (K-2)—or $\text{ANGSHL}$ (L-21d), for GCP input—for the corresponding layer.

T-4 is repeated $\text{NT4}$ times (H-1).
Specify the cone generator, \( x \), for \( \mathbf{r}_w \):

\[
x = -\zeta_g \quad \therefore \quad \mathbf{r}_w = (0, 0, -1)
\]

\( x_w \) = projection of \( \mathbf{r}_w \) onto element plane

\( \zeta = \text{ZETA} \) is a right-handed rotation about \( \zeta \), the fabrication normal.

**Figure 8.3** Specifying the Wall-Reference Vector, \( \mathbf{r}_w \), in an Element Unit: example showing use of a conical-shell generator
<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
<th>KELT</th>
<th>IWALL</th>
<th>ZETA</th>
<th>ECZ</th>
<th>ILIN</th>
<th>IPLAS</th>
<th>INTEG</th>
<th>IPENL</th>
<th>IANG</th>
<th>USERELT</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>node 1; see Section 14.5 “Shell and Mesh-Transition Shell Elements” on page 14-19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>node 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>node 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N4</td>
<td>node 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KELT</td>
<td>element code number (see Chapter 14 “The Element Library”)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IWALL</td>
<td>wall fabrication identifier:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 – shell wall properties are given in user-written subroutine WALL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&gt;0 – shell wall configuration (fabrication) number in the Wall Fabrication Table (K-1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;0 – shell wall fabrication identifier in the GCP Fabrications Table (l-21a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>When IWALL = 0, data below indicated by ✤ are defined in subroutine WALL; they are automatically initialized to zero before WALL is called, thereby superseding any nonzero values input here</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>✤ ZETA</td>
<td>angle ( \zeta ) between the wall-reference coordinate ( x_w ) and the fabrication coordinate ( x ); ( \zeta ) is a right-handed rotation about ( x ); see Figure 8.2 on page 8-19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>✤ ECZ</td>
<td>eccentricity in ( z' ) direction. ECZ is the ( z' ) coordinate of the shell wall middle surface; refer to Figure 6.2 on page 6-28; in element units, ( (x', y', z') ) are used in place of ( (X', Y', Z') ), which do not exist in an element unit; see “Effects of Eccentricity” on page 16-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>✤ ILIN</td>
<td>governs geometric nonlinearity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 – nonlinear strain-displacement relations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 – linear strain-displacement relations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Note that with ILIN = 1, bifurcation buckling is suppressed in the shell unit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>✤ IPLAS</td>
<td>governs material nonlinearity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 – elastic behavior only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 – plasticity included, with the material law satisfied at each element integration point</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 – plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**INTEG**

integration type (see N-1)

0 – standard integration, or \(2 \times 2\) Gauss points for elements **E410** & **E411**

1 – modified 5-point integration, previously referred to as full integration

2 – full \(3 \times 3\) Gauss integration for **E411** element

**IPENL**

penalty option (see N-1)

0 – no penalty function on fourth-order terms in elements **E410** and **E411**

1 – penalty function included in elements **E410** and **E411**

**IANG**

wall-reference option; see discussion on page 8-18

0 – use default strategy of projecting \(x_g, y_g\) to establish \(x_w\)

1 – input \(r_w\), which is projected to establish \(x_w\); see (T-4b)

**USERELT**

user-specified element number, used only when \(IUWLE = 1\) on H-1

\[
\text{if } ( \text{KELT references an element with > 4 nodes }) \text{then go to T-4a}
\]

\[
\text{elseif } ( \text{IANG = 1} ) \text{ then go to T-4b}
\]

\[
\text{else follow instructions at end of T-4c}
\]
**T-4a Extra Nodes**

This record is provided only for elements having more than four nodes, and is used to input nodes 5–n, where n is the total number of nodes in the element. The order in which the nodes are given is according to the numbering shown in the sketches in Section 14.5 “Shell and Mesh-Transition Shell Elements”. Some 4-node elements, like the E411, contain midside deviational nodes that are added automatically without user input. Currently, only the 9-node E480 element requires a T-4a record.

<table>
<thead>
<tr>
<th>NODE(i), i=5,n</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE(i)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IANG (T-4)</th>
<th>wall-reference option</th>
</tr>
</thead>
<tbody>
<tr>
<td>if</td>
<td>( IANG = 1 ) then go to T-4b</td>
</tr>
<tr>
<td>else</td>
<td>follow instructions at end of T-4b</td>
</tr>
</tbody>
</table>
**T-4b Wall Reference Vector**

**RX, RY, RZ**

Wall reference vector, \( \mathbf{r}_w \), which is projected onto the element surface to determine the direction of the wall-reference coordinate \( x_w \); \( \mathbf{r}_w \) is expressed in \((x_g', y_g', z_g')\) global coordinates.

- **NT4** (H-1) number of quadrilateral shell elements
- **NT5** (H-1) element-command flag

If (fewer than **NT4** quadrilaterals defined) return to **T-4**

Elseif (\( \text{NT5} = 1 \)) then go to **T-5**

Elseif (\( \text{NT5} = 2 \)) then go to **T-5**

Elseif (\( \text{NT5} = 3 \)) then go to **T-100**

Else go to **U-1**
8.5 Definition of “Other” Elements via the Edef Protocol

If and only if the NT5 parameter is 1 or 2 on the H-1 record for the current element unit, STAGS expects the analyst to continue using the Edef protocol to define zero or more type E810, E820 and/or E822 contact elements, then to define zero or more type E830, E840, E845, E847 and/or E849 sandwich elements, and then to define zero or more E880-family solid elements via additional type T-x and/or T-xx “regular” input records, as described below. A type T-5 control record is required next to specify how many definitions are to be made for each of these types or classes of elements.

T-5 Contact, Sandwich, and Solid Element Flags

This record is required next (with the Edef protocol) for specification of the number of “regular” input definitions that STAGS must process to generate zero or more E810 (PAD) elements, then to generate zero or more E820 (generalized contact) elements, then to generate zero or more E822 (line-contact-interaction) definitions, then to generate zero or more E830, E840, E845, E847 and/or E849 (sandwich) elements, and finally to generate zero or more E880-family (ANS-type solid) elements.

Each type E810 PAD element to be generated may be defined explicitly with a single T-6 record, which references eight user points that must have been defined with S-1/S-1a/S-2 or S-3/S-3a/S-4 user point records within that element unit. Multiple PAD elements can be defined explicitly with a T-6 record on which the PAD-element looping parameter (described below) is invoked, using nodal incrementations specified on a companion T-6b record.

Specific type E820 “general-contact” elements in an element unit are constructed on-the-fly by the STAGS program when contact occurs during the course of an analysis—utilizing “contact-definition” information supplied here. Each contact definition identifies a set of one or more contact points on one structural component that may come into contact with a specific contact surface on another. Each such contact definition is specified with a single T-7 record set (consisting of a T-7a record followed by as many T-7b or T-7c contact-surface-element-specification records and by as many T-7d or T-7e contact-point-specification records as may be required), as described below.

Similarly, specific line-contact elements are constructed on-the-fly by STAGS when edge-to-edge contact occurs during the course of an analysis—using contact lines that have been specified as
described earlier (with the S-5 and related records) and utilizing “line-contact-interaction-definition” information that the analyst must supply here. Each line-contact-interaction definition identifies a specific line on one structural component that may be contacted by a specific contacting line on another. Each such line-contact-interaction definition is specified with a single T-8 record.

Each E810-element-, each E820-element- and each E822-element specification references a stiffness/displacement (penalty) function table that must have been defined via l-4a, l-4b, l-4c and l-4d records.

Each individual E830 sandwich element in a STAGS element unit may be defined via its own T-9/T-9a/T-9c/T-9e record set (plus T-9b, T-9d and/or T-9f records, if material-angle transformations are required). Multiple E830 sandwich elements can be defined using the two looping parameters on the T-9 record, with nodal incrementations specified on the T-9g and T-9h companion records.

Each individual E840 sandwich element in a STAGS element unit may be defined via its own T-10/T-10a/T-10c/T-10e record set (plus T-10b, T-10d and/or T-10f records, if material-angle transformations are required). Multiple E840 sandwich elements can be defined via the two looping parameters on T-10, with nodal incrementations specified on T-10g and T-10h.

Each individual E845 sandwich transition element in a STAGS element unit may be defined via its own T-11/T-11a/T-11c/T-11e record set (plus T-11b, T-11d and/or T-11f records, if material-angle transformations are required). Multiple E845 elements can be defined via the two looping parameters on T-11, with nodal incrementations specified on T-11g and T-11h.

Each individual E847 sandwich transition element in a STAGS element unit may be defined via its own T-12/T-12a/T-12c/T-12e record set (plus T-12b, T-12d and/or T-12f records, if material-angle transformations are required). Multiple E847 elements can be defined via the two looping parameters on T-12, with nodal incrementations specified on T-12g and T-12h.

Each individual E849 sandwich element in a STAGS element unit may be defined via its own T-13/T-13a/T-13c/T-13e record set (plus T-13b, T-13d and/or T-13f records, if material-angle transformations are required). Multiple E849 elements can be defined via the two looping parameters on T-13, with nodal incrementations specified on T-13g and T-13h.

Each individual E880-family solid element in a STAGS element unit may be defined explicitly via its own T-14/T-14a record set (plus a T-14b record, if a material-angle transformation is required). Multiple E880-family solid elements can be defined via the two looping parameters on T-14, with nodal incrementations specified on T-14c and T-14d.
The T-5 record contains three contact-element-definition flags ($N_{810}$, $N_{820}$ and $N_{822}$), five sandwich-element-definition flags ($N_{830}$, $N_{840}$, $N_{845}$, $N_{847}$ and $N_{849}$), and one solid-element-definition flag ($N_{880}$):

| $N_{810}$ | number of PAD-element-definition records for the current element unit |
| $N_{820}$ | number of general-contact-definition records |
| $N_{822}$ | number of line-contact-interaction definition records |
| $N_{830}$ | number of 6-node-sandwich-element-definition records |
| $N_{840}$ | number of 8-node-sandwich-element-definition records |
| $N_{845}$ | number of 10-node-sandwich-transition-element-definition records |
| $N_{847}$ | number of 14-node-sandwich-transition-element-definition records |
| $N_{849}$ | number of 18-node-sandwich-element-definition records |
| $N_{880}$ | number of solid-element-definition records |

if ($N_{810} > 0$) then go to T-6
elseif ($N_{820} > 0$) then go to T-7
elseif ($N_{822} > 0$) then go to T-8
elseif ($N_{830} > 0$) then go to T-9
elseif ($N_{840} > 0$) then go to T-10
elseif ($N_{845} > 0$) then go to T-11
elseif ($N_{847} > 0$) then go to T-12
elseif ($N_{849} > 0$) then go to T-13
elseif ($N_{880} > 0$) then go to T-14
else go to U-1
Contact elements

The current version of STAGS has three “elements” that enable the user to perform analyses in which three types of contact might occur.

The E810 PAD surface/surface contact “element” is designed to treat situations in which one surface may come into contact with another, when the specific elements on one surface that may come into contact with specific elements on the other surface are known \textit{a priori}.

The E820 generalized contact “element” is designed to treat situations in which one surface may come into contact with another surface, but where the specific elements that may come into contact with each other are \textit{not} known in advance and/or when they may change during the course of the analysis to be performed. The E820 approach considers contact to be point/surface phenomenon. Here, the user specifies a set of points on one surface and a set of elements on the other, telling STAGS that one or more of the identified points \textit{may} come into contact one or more of the identified elements. STAGS digests this information and uses it to determine whether or not point/surface contact has started (or is continuing) at each step of the analysis, and generates one or more point/surface contact elements as and if required, for the specific points and surfaces that are involved with that contact.

The E830 line/line contact “element” is designed to treat situations in which one line (edge) may come into contact with another line (edge), but where the exact location at which that contact may occur is not known in advance and/or when it may change during the course of the analysis. The E830 approach considers this type of contact to be line/line phenomenon. Here, the user specifies a set of two or more line segments in the model—and uses an E830 line/line contact “element” to identify one line that \textit{may} come into contact with another, for each line/line contact situation that might occur. STAGS digests this information and uses it to determine whether or not line/line contact has started (or is continuing) at each step of the analysis, and generates one or more line/line contact elements as and if required.

The theoretical foundations of these three types of contact “elements” are described in Section 14.8 of this document and in the \textit{STAGS Elements Manual} document. The input requirements for these elements are described next.
**T-6 E810 PAD Contact Element**

The **E810** 8-node **PAD** contact element is basically a set of four independent nonlinear springs connecting the nodes defining one **E410** shell element to the corresponding nodes of a second **E410** element. It is intended for use in situations where those two PAD-connected elements—which may be in the same or in different shell or element units—*may* come into contact with each other. The **E810** PAD element is suitable for use only in situations where the contact region is known *a priori*, where the individual elements coming into contact with each other can be readily identified and paired, and where no *sliding* along the contact surface occurs. The PAD element can be used to treat (some) lap-joint and Hertzian impact problems, but it is *not* designed for use in more general situations where the contact regions are not known *a priori* or when sliding occurs or when friction is present. **STAGS'** more general (**E820**) contact capabilities must be used in those situations.

![Figure 8.4 E810 PAD Element](image)

The nonlinear spring connecting each pair of nodes typically has a very low stiffness when the *gap* separating the two nodes is positive (where the gap is defined as the distance of the upper-element node from the lower-element reference plane) and to be very stiff when the gap is extremely small or negative (indicating that at least some portions of the two parent elements are in contact with each other): the large forces and stiffnesses that are produced by stiff PAD-element springs helps to enforce the displacement compatibility constraints for the contact problem.

Stiffness properties for PAD-element springs must be specified *via* one or more stiffness-displacement tables (type l-4a through l-4d records). These stiffness profiles, each of which is
identified by its *profile-definition table number*, are currently assumed to be independent of velocity (any velocity dependencies specified are ignored by the program).

Element-unit *user point* nodes employed in defining PAD elements are defined *via* type S-1 or S-3 records (and type S-2 or S-4 records, if necessary): these *user points* are generally slaved to nodes defining the E410 shell elements that are expected to come in contact with each other. An additional *OFFSET* parameter can be used to account for thickness effects when the user point nodes lie on parent-node reference surfaces within (rather than on the surfaces of) those parent elements.

The T-6 record defines one or more E810 elements to be included in the current element unit. The *looping* capabilities provided here, *via* NX and on T-6 and the incrementation parameters on T-6a, can be used effectively in many situations.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
<th>N5</th>
<th>N6</th>
<th>N7</th>
<th>N8</th>
<th>KELT</th>
<th>ITAB</th>
<th>OFFSET</th>
<th>NX</th>
<th>USERELT</th>
</tr>
</thead>
</table>
| N1 | first lower-surface pad-element node point  
N2 | second lower-surface pad-element node point  
N3 | third lower-surface pad-element node point  
N4 | fourth lower-surface pad-element node point  
N5 | first upper-surface pad-element node point, connected to N1  
N6 | second upper-surface pad-element node, connected to N2  
N7 | third upper-surface pad-element node, connected to N3  
N8 | fourth upper-surface pad-element node, connected to N4  
KELT | element code number 810 defines an 8-node PAD element  
ITAB | spring stiffness-displacement-table identifier (see record I–4a)  
OFFSET | offset parameter, to account for element thicknesses  
NX | looping parameter, set equal to unity by STAGS if omitted or nonpositive  
USERELT | user-specified element number, used only if IUWLE = 1 on H-1 |

if (NX > 1) go to T-6a
else follow instructions at the end of T-6a
**T-6a PAD Element Incrementations**

If the \( nx \) looping parameter is greater than 1 on T-6, a T-6a record is required to specify nine incrementation variables to be used with the T-6 looping function.

<table>
<thead>
<tr>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>I4</th>
<th>I5</th>
<th>I6</th>
<th>I7</th>
<th>I8</th>
<th>I9</th>
</tr>
</thead>
<tbody>
<tr>
<td>incrementation for the N1 (PAD node) variable on the parent T-6 record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation for the N2 (PAD node) variable on the parent T-6 record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation for the N3 (PAD node) variable on the parent T-6 record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation for the N4 (PAD node) variable on the parent T-6 record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation for the N5 (PAD node) variable on the parent T-6 record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation for the N6 (PAD node) variable on the parent T-6 record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation for the N7 (PAD node) variable on the parent T-6 record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation for the N8 (PAD node) variable on the parent T-6 record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation for IUWLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero, or positive, as required.

**Example:** the following T-6/T-6a record combination:

```
   10 20 30 40 50 60 70 80 810 7 0.0 3
   1 1 1 1 1 1 1 1
```

generates the same three PAD elements as the following three individual T-6 records:

```
   10 20 30 40 50 60 70 80 810 7 0.0
   11 21 31 41 51 61 71 81 810 7 0.0
   12 22 32 42 52 62 72 82 810 7 0.0 1
```
| N810 (T-5) | number of E810 definitions |
| N820 (T-5) | number of E820 definitions |
| N822 (T-5) | number of E822 definitions |
| N830 (T-5) | number of E830 definitions |
| N840 (T-5) | number of E840 definitions |
| N845 (T-5) | number of E845 definitions |
| N847 (T-5) | number of E847 definitions |
| N849 (T-5) | number of E849 definitions |
| N880 (T-5) | number of E880 definitions |

if (N810 E810 definitions have been processed) return to T-6
elseif (N820 > 0) then go to T-7
elseif (N822 > 0) then go to T-8
elseif (N830 > 0) then go to T-9
elseif (N840 > 0) then go to T-10
elseif (N845 > 0) then go to T-11
elseif (N847 > 0) then go to T-12
elseif (N849 > 0) then go to T-13
elseif (N880 > 0) then go to T-14
else go to U-1
T-7 General Contact Definition

As noted in the preceding description, E810 PAD elements are defined explicitly and should only be used in situations where particular E410 elements that may contact each other are known a priori and can be paired, and where the contact regions (and E410-element pairings) do not change during an analysis. STAGS’ more general point/surface contact capabilities must be used in situations where contact is anticipated but it is not known where the contact will occur and/or how the contact region changes as the analysis progresses.

It is convenient, here, for the analyst to view the contact problem as one in which one structure is (perhaps arbitrarily) designated as the contacting structure and the other is considered to be the contacted one. The current approach in STAGS to the general contact problem avoids many of the complexities and inefficiencies of general surface-on-surface interactions by considering one or more specific points on one structure that may come into contact with one or more shell elements on the other structure—as shown schematically in Figure 8.5.

![Figure 8.5 Contact-Definition Specification in STAGS](image)

The general-contact capabilities in STAGS are invoked when the analyst, anticipating the possibility that contact between two structural components may occur, includes one or more E820 contact-definition specifications in an element unit of the STAGS model. Each contact definition identifies a set of contact points on the contacting structure that may experience contact with a particular contact surface on the contacted structure—where the contact surface is a set of contiguous shell elements forming a convex region on the surface of the latter body.
STAGS uses these *contact-definitions* to check for contact and to construct actual contact elements coupling contacting points with contacted shell elements on-the-fly, as and if required, as the analysis progresses. In doing this, STAGS attempts to use penalty functions to enforce displacement-compatibility constraints between each contacting point and each element with which it is in contact—utilizing analyst-supplied *stiffness-vs.-displacement* information to compute the forces and stiffnesses resulting from the (necessarily) small contact-surface penetrations that occur.

An actual contact element is (conceptually) a nonlinear spring connecting the contacting point to the surface of the contacted element. This nonlinear spring typically has a very low stiffness and generates a small force when the *contact-surface* penetration is small, but it gets progressively stiffer and generates a larger force as the penetration increases. Stiffness properties for these springs must be specified *via* one or more stiffness-displacement tables (type I-4a through I-4d records). These stiffness-displacement functions each of which is identified by its *table number*, are currently assumed to be independent of velocity (any velocity dependencies specified are ignored by the program).

The T-7 record defines a single **E820** definition to be included in the current element unit. An **E820 contact-definition** is made *via* a T-7 record (described immediately below) that is followed first by as many T-7a or T-7b *contact-surface-element-specification* records, and then by as many T-7c or T-7d *contact-point-specification* records as may be required—as described farther below. Each of these *contact-surface-specifications* references a specific stiffness-displacement (penalty) function table.
Definition of “Other” Elements via the Edef Protocol

Model Input—Element Units (2)

STAGS 5.0 User Manual April, 2009 8-35

---

**KELT  NSRF  NPTS**

**KELT** element code number 820 identifies a contact-definition specification

**NSRF** parameter indicating the method to be used to identify the contiguous shell elements that comprise the *contact surface*: set $\text{NSRF} \geq 0$ to use the *row & column* method (to identify one or more elements in one or more shell units), or set $\text{NSRF} < 0$ to use the *element-number* method (to identify one or more elements in one or more shell units and/or one or more elements in the current or previously-defined element units)

**NPTS** parameter indicating the method to be used to identify the specific points that *may* come into contact with the *contact surface*: set $\text{NPTS} \geq 0$ to use the *row & column* method (to identify one or more nodes in one or more shell units), or set $\text{NPTS} < 0$ to use the *point-number* method (to identify one or more nodes in one or more shell units and/or one or more nodes in the current or previously-defined element units)

\[
\text{if} \quad ( \text{NSRF} \geq 0 ) \quad \text{then} \quad \text{go to T-7a}
\]

\[
\text{else} \quad \text{go to T-7b}
\]
**T-7a E820 Row & Column Contact-Element Specifications**

**NSRF ≥ 0 (T-7)** indicates that the row & column method is to be used to identify the contiguous shell elements that comprise the contact surface. This method clearly can only be used to identify contact elements that are associated with one or more shell units, because there are no row or column associations for elements in element units. With this method, the value of the **NSRF** parameter indicates the minimum number of contact elements to be identified with the set of one or more T-7a records included for the current contact-definition. The number of contact elements identified by any given T-7a depends on the parameters on that record—as described below:

USRF TYPE LI LJ ID NI NJ

**USRF** identifies the shell unit within which the element(s) identified by the **LI** and **LJ** row & column parameters are defined; the element grid for this shell unit has **NROWS** rows and **NCOLS** columns of quadrilateral domains—each of which contains one **E410** quadrilateral element or two **E320** triangular elements.

**TYPE** specifies the element type for this (or these) element(s); **TYPE** must currently be either **E410** or **E320**.

**LI**, **LJ** row and column numbers identifying one or more elements: **LI = LJ = 0** indicates that all type **TYPE** elements of shell unit **USRF** are to be included; **LI > 0** with **LJ = 0** identifies (and includes) all of the elements in row # **LI** of the element grid; **LJ > 0** with **LI = 0** identifies all of the elements in column **LJ**; and **LI > 0** with **LJ > 0** identifies one or more **E410** element or the one or more pairs of **E320** elements, starting at row **LI** and column **LJ** of the unit: if **NI** and **NJ** are absent, zero or unity, a single element (or pair of elements) is defined; if **NI** and **NJ** are both positive, then **NIxNJ** elements (or pairs of elements) are identified in a double FORTRAN-like loop, starting with the element (or pair of elements) at (**LI**, **LJ**).

**ID** spring stiffness-displacement table identifier (see record I–4a)

**NI**, **NJ** element-identification looping parameters, as discussed in **LI**, **LJ**, above

- **NSRF** (T-7) surface-method parameter
- **NPTS** (T-7) point-method parameter

if ( fewer than **NSRF** contact-surface elements have been identified ) return to **T-7a**
elseif( **NPTS** > 0 ) then go to **T-7c**
else go to **T-7d**
**T-7b E820 Element-Number Contact-Element Specifications**

\( \text{NSRF} \leq 0 \) (T-7) indicates that the *element-number method* is to be used to identify the contiguous shell elements that comprise the *contact surface*. This method can (but generally should not) be used to identify contact elements that are associated with *shell units*, and it *must* be used to identify elements that are defined in STAGS *element units*. With this method, the *magnitude* of the \( \text{NSRF} \) parameter indicates the *minimum number* of contact elements to be identified with the set of one or more T-7b records included for the current *contact-definition*.

The number of contact elements identified by any given T-7b depends on the parameters on that record—as described below.

<table>
<thead>
<tr>
<th>USRF</th>
<th>TYPE</th>
<th>I1</th>
<th>I2</th>
<th>INC</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>USRF</td>
<td>identifies the <em>shell or element unit</em> within which the element(s) identified by the ( I_1, I_2 ) and ( \text{INC} ) parameters are defined</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TYPE</td>
<td>specifies the element type for this (or these) element(s): ( \text{TYPE} ) must currently be either ( \text{E410} ) or ( \text{E320} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( I_1, I_2, \text{INC} )</td>
<td>element identifiers: ( I_1 &gt; 0 ), with ( I_2 = 0 ) and ( \text{INC} = 0 ) identifies element # ( I_1 ) (in unit ( \text{USRF} )) as part of the contact surface for the current contact-definition; ( I_1, I_2, \text{INC} &gt; 0 ) identifies ( I_2 ) elements—starting with element ( I_1 ) and incrementing by ( \text{INC} ) until ( I_2 ) elements have been indicated (in unit ( \text{USRF} )) as part of the <em>contact surface</em> for the current <em>contact-definition</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>spring stiffness-displacement table identifier (see record l–4a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{NSRF} \] (T-7) *surface-method parameter*  
\[ \text{NPTS} \] (T-7) *point-method parameter*

If (fewer than \( \text{NSRF} \) contact-surface elements have been identified )\textit{return to} \( \text{T-7b} \)  
Elseif (\( \text{NPTS} > 0 \)) then \textit{go to} \( \text{T-7c} \)  
Else \textit{go to} \( \text{T-7d} \)
**T-7c Row & Column Contact-Point Specifications**

**NPTS ≥ 0** (T-7) indicates that the *row & column method* is to be used to identify the points that may come into contact with the *contact surface* specified for the current *contact-definition*. This method clearly can only be used to identify nodes that are associated with one or more *shell* units, because there are no row or column associations for nodal points in element units. With this method, the value of the **NPTS** parameter indicates the *minimum number* of contact points to be identified with the set of one or more T-7c records included for the current *contact-definition*. The number of contact points identified by any given T-7c depends on the parameters on that record—as described below:

<table>
<thead>
<tr>
<th>UNITP</th>
<th>LI</th>
<th>LJ</th>
<th>RADIUS</th>
<th>TOUCHE</th>
<th>NI</th>
<th>NJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNITP</td>
<td>identifies the <em>shell unit</em> within which the contact point(s) identified by the LI and LJ row &amp; column parameters are defined; the nodal mesh for this shell unit has NROWS rows and NCOLS columns of nodal points</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LI,LJ</td>
<td>row and column numbers identifying one or more contact points: LI = LJ = 0 indicates that all of the node points of shell unit UNITP are to be considered as contact points; LI &gt; 0 with LJ = 0 identifies all of the nodes in row LI of the nodal grid; LJ &gt; 0 with LI = 0 identifies all of the nodes in column LJ; and LI &gt; 0 with LJ &gt; 0 identifies one or more nodes, starting at row LI and column LJ of the unit: if NI and NJ are absent, zero or unity, a single point (LI,LJ) is defined; if NI and NJ are both positive, then NIxNJ points are identified in a double FORTRAN-like loop, starting with the point at (LI,LJ).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RADIUS</td>
<td>specifies a radial offset parameter to be used in determining whether or not the point(s) identified by the current T-7c are in contact with the <em>contact surface</em>; this parameter can be used to account for the thickness of material surrounding the contact point(s) in much the same way as the thicknesses of contact elements on the <em>contact surface</em> are taken into account</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOUCHE</td>
<td>provides information required if the identified contact point(s) are in contact with the <em>contact surface</em> at the outset of the problem: set TOUCHE = 0 if not; or set TOUCHE = 1 if the point(s) are contacting the surface from its <em>positive</em> side; or set TOUCHE = -1 if the point(s) are contacting the surface from its <em>negative</em> side—where the normal vector for the <em>contact surface</em> points out of the <em>positive</em> side of the surface and away from the <em>negative</em> side</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NI,NJ</td>
<td>point-identification looping parameters, as discussed in LI,LJ, above</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

if (fewer than NPTS contact-points have been identified) go to T-7c  
else follow instructions at the end of T-7d
**T-7d Point-Number Contact-Point Specifications**

*NPTS* < 0 (on T-7) indicates that the *point-number method* is to be used to identify the nodal points that may come into contact with the *contact surface*. This method can (but generally should not) be used to identify nodes that are associated with *shell units*, and it *must* be used to identify points that are defined in STAGS *element units*. With this method, the *magnitude* of the *NPTS* parameter indicates the *minimum number* of contact points to be identified with the set of one or more T-7d records included for the current *contact-definition*. The number of *contact points* identified by any given T-7d depends on the parameters on that record—as described below.

<table>
<thead>
<tr>
<th>UNITP</th>
<th>I1</th>
<th>I2</th>
<th>INC</th>
<th>RADIUS</th>
<th>TOUCHE</th>
</tr>
</thead>
</table>

- **UNITP** identifies the *shell or element unit* within which the *contact point(s)* identified by the *I1, I2 and INC* parameters are defined; the nodal mesh for this shell unit has *NROWS* rows and *NCOLS* columns of node points.

- **I1, I2, INC** point identifiers: *I1 > 0*, with *I2 = INC = 0* identifies point # *I1* (in unit *UNITP*) as a *contact point* for the current *contact-definition*; *I1, I2, INC > 0* identifies *I2* points—starting with point *I1* and incrementing by *INC* until *I2* points have been indicated (in unit *UNITP*) as *contact points* for the current *contact-definition*.

- **RADIUS** specifies a radial offset parameter to be used in determining whether or not the point(s) identified by the current T-7 definition are in contact with the *contact surface*: this parameter can be used to account for the thickness of material surrounding the *contact point(s)* in much the same way as the thicknesses of contact elements on the *contact surface* are taken into account.

- **TOUCHE** provides information required if the identified *contact point(s)* are in contact with the *contact surface* at the outset of the problem: set *TOUCHE = 0* if not; or set *TOUCHE = 1* if the point(s) are contacting the surface from its *positive side*; or set *TOUCHE = −1* if the point(s) are contacting the surface from its *negative side*—where the normal vector for the *contact surface* points out of the *positive side* of the surface and away from the *negative side*. 
N820 (T-5)  number of E820 definitions
N822 (T-5)  number of E822 definitions
N830 (T-5)  number of E830 definitions
N840 (T-5)  number of E840 definitions
N845 (T-5)  number of E845 definitions
N847 (T-5)  number of E847 definitions
N849 (T-5)  number of E849 definitions
N880 (T-5)  number of E880 definitions

if ( fewer than NPTS contact-points have been identified ) return to T-7d
elseif ( fewer than N820 E820 definitions have been processed ) return to T-7
elseif ( N822 > 0 ) then go to T-8
elseif ( N830 > 0 ) then go to T-9
elseif ( N840 > 0 ) then go to T-10
elseif ( N845 > 0 ) then go to T-11
elseif ( N847 > 0 ) then go to T-12
elseif ( N849 > 0 ) then go to T-13
elseif ( N880 > 0 ) then go to T-14
else go to U-1
## T-8 Line-Contact Interaction Definition

The T-8 record specifies a pair of “contact” lines (which must have been defined via S-5 records, as described on page 7-17) that may experience line-to-line contact with each other during the course of the analysis that STAGS is to perform. The first line of each such pair must have one or more type E410 elements associated with it (qualifying it to be a contacted line), as described earlier. The second line, which may but does not need to have elements associated with it, is treated as the contacting line. When contact occurs at one or more points along the two lines, STAGS uses penalty information supplied by the analyst to generate stiffnesses and forces on the fly to treat that contact.

<table>
<thead>
<tr>
<th>LINE1</th>
<th>LINE2</th>
<th>IPEN</th>
<th>NX</th>
<th>INC1</th>
<th>INC2</th>
<th>INC3</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINE1</td>
<td>identifies the contacted line, which must have one or more elements associated with it; this line must have been specified via an S-5 record, as described on page 7-17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LINE2</td>
<td>identifies the contacting line, which may (but is not required to) have one or more elements associated with it; this line must also have been specified via an S-5 record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPEN</td>
<td>identifies the penalty function table (of stiffness as a function of penetration) that STAGS is to use in calculating stiffnesses and forces that arise when contact occurs between these two lines; the table identified must have been specified via an I-4 record set, as described above</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NX</td>
<td>indicates the number of line contact interactions that are to be specified with information on this T-8 record; STAGS sets NX = 1 if it is not positive</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC1</td>
<td>incrementation parameter for LINE1 (used only when NX &gt; 0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC2</td>
<td>incrementation parameter for LINE2 (used only when NX &gt; 0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC3</td>
<td>incrementation parameter for IPEN (used only when NX &gt; 0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
if ( fewer than \text{N822}  \text{E822}  \text{E822} definitions have been processed ) \text{return to } T-8

elseif ( \text{N830} > 0 ) \text{ then } \text{go to } T-9

dot

else \text{go to } U-1
Sandwich elements

Sandwich structures play such an important role in the design of many aerospace structures, so it is necessary that their behavior be determined adequately. The classical approach to the modeling of sandwiches is an extension of thin-shell-theory, in which the behavior of a three-dimensional sandwich object is reduced to the behavior of a two-dimensional surface with in-and out-of-plane stiffness properties. In some programs, a sandwich construction is idealized as a pair of membranes that are held apart by a core that has a relatively large resistance against transverse shear, is virtually inextensional in the transverse direction, and has virtually no stiffness in the middle-surface plane. This type of model is called a sandwich of the first kind. Finite element discretizations of this kind of classical model have been made in the past, but successful applications have been confined primarily to linear or moderately nonlinear analyses. A more general sandwich model emerges by considering the two faces to be shells (with bending as well as the usual membrane stiffness) that are held apart by a lightweight core. The core may have three dimensional elastic properties. This type of model is called a sandwich of the second kind, and is the model that is implemented in STAGS.* A good example of a problem requiring this model would be one with structural walls made of glass fiber faces and using polyurethane foam as the core material.

The current version of STAGS has three “standard” sandwich elements (element types E830, E840 and E849) and two mesh-transition sandwich elements (E845 and E847). STAGS input requirements for each of these are described next.

* See, for example:
The 6–node **E830** sandwich element in **STAGS** is a construction of two triangular-shell face elements that are held apart by a lightweight core, as shown in Figure 8.6. The core may have three dimensional elastic properties.

**Figure 8.6** E830 6-Node Sandwich Element

Definition of one or more 6–node **E830** sandwich elements is accomplished with the following set of records:

- **T-9** to specify parameters used for all parts of the element
- **T-9a** to specify information for the lower face sheet of the element
- **T-9b** to specify lower face sheet transformation angles (optional)
- **T-9c** to specify information for the upper face sheet
- **T-9d** to specify upper face sheet transformation angles (optional)
- **T-9e** to specify core parameters
- **T-9f** to specify core transformation angles (optional)
- **T-9g** for “x-direction” incrementation parameters
- **T-9h** for “y-direction” incrementation parameters

These are described sequentially in the following text.
**KELT**    element code number (must be 830)

**ILIN**    geometric-linearity flag:

- 0  – nonlinear strain-displacement relations
- 1  – linear strain-displacement relations

**INTEG**    number of surface-integration points: set **INTEG** = 1, 3, 4 or 7; if **INTEG** = 0, **STAGS** sets **INTEG** = 3

**IPEN**    penalty option (see N-1)

- 0  – no penalty function on fourth-order terms in **E330** elements
- 1  – penalty function included in **E330** elements

**NX**    x-direction looping parameter, set equal to unity by **STAGS** if omitted or nonpositive; **NX** > 1 instructs **STAGS** to generate a set of **nx** type **E830** elements in the x direction, using nodal incrementation information given on the T-9g record

**NY**    y-direction looping parameter, set equal to unity by **STAGS** if omitted or nonpositive; **NY** > 1 instructs **STAGS** to generate a set of **NY** type **E830** elements in the y direction for each of the **NX** elements generated in the x-direction in the current definition, using nodal incrementation information given on the T-9h record

**USERC**    user-specified element number for core component

(used only if **IUWLE** = 1 on H-1)

**USER1**    user-specified element number for **lower face sheet**, (if **IUWLE** = 1 on H-1)

**USER2**    user-specified element number for **upper face sheet**, (if **IUWLE** = 1 on H-1)

⚠️ go to T-9a
### T-9a E830 Lower Face-Sheet Properties

A single T-9a record must be included immediately following the T-9 record, to specify the nodes for the lower face sheet of the first E830 element and to specify other parameters for the lower face sheet of each of the element(s) specified in the current definition.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>IFABL</th>
<th>ZETAL</th>
<th>ECZL</th>
<th>IPLASL</th>
<th>IANGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td></td>
<td></td>
<td>first lower face sheet node point</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td></td>
<td></td>
<td>second lower face sheet node point</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td></td>
<td></td>
<td>third lower face sheet node point</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**IFABL**
- fabrication identifier for the lower face sheet element(s); this is equivalent to the **WALL** parameter for E330 triangular elements:
  - >0 – wall configuration number in the Wall Fabrication Table (K-1)
  - 0 – shell wall properties are given in user-written subroutine WALL (See the T-4 record description for more information about this)
  - <0 – shell fabrication identifier in the GCP Fabrication Table (I-21a)

**ZETAL**
- angle $\zeta$ between the wall-reference coordinate $x_w$ and the fabrication coordinate $\bar{x}$, for the lower face sheet; $\zeta$ is a right-handed rotation about $\bar{z}$; see Figure 8.2 on page 8-19.

**ECZL**
- eccentricity in $z'$ direction, for the lower face sheet; **ECZL** is the $z'$ coordinate of the shell wall middle surface; please refer to Figure 6.2 on page 6-28; in element units, $(x', y', z')$ are used in place of $(X', Y', Z')$, which do not exist in an element unit; see “Effects of Eccentricity” on page 16-6.

**IPLASL**
- lower face sheet material-nonlinearity flag:
  - 0 – elastic behavior only
  - 1 – plasticity included, with the material law satisfied at each element integration point
  - 2 – plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)
IANGL  lower face sheet wall-reference option; see discussion on page 8-18

0  – use default strategy of projecting $x_g,y_g$ to establish $x_w$

1  – input $r_w$, which is projected to establish $x_w$; see (T-9b)

\[
\text{if ( IANGL > 0 ) then} \quad \text{go to T-9b}
\]
\[
\text{else} \quad \text{go to T-9c}
\]
**T-9b E830 Lower Face-Sheet Wall Reference Vector**

**RXL RYL RZL**

RXL, RYL, RZL lower face sheet wall reference vector, \( \mathbf{r}_w \), which is projected onto the element surface to determine the direction of the wall-reference coordinate \( x_w \); \( \mathbf{r}_w \) is expressed in \( (x_g, y_g, z_g) \) global coordinates.

*go to T-9c*
**T-9c E830 Upper Face-Sheet Properties**

A single T-9c record must be included immediately following the T-9a (or T-9b) record, to specify the nodes for the upper face sheet of the first E830 element and to specify other parameters for the upper face sheet of each of the element(s) specified in the current definition.

<table>
<thead>
<tr>
<th>N4</th>
<th>N5</th>
<th>N6</th>
<th>IFABU</th>
<th>ZETAU</th>
<th>ECZU</th>
<th>IPLASU</th>
<th>IANGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>N4</td>
<td>first upper face sheet node point</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N5</td>
<td>second upper face sheet node point</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N6</td>
<td>third upper face sheet node point</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFABU</td>
<td>fabrication identifier for the upper face sheet element(s); this is equivalent to the WALL parameter for E330 quadrilateral elements:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;0</td>
<td>wall configuration number in the Wall Fabrication Table (K-1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>shell wall properties are given in user-written subroutine WALL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;0</td>
<td>shell fabrication identifier in the GCP Fabrication Table (I-21a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZETAU</td>
<td>angle $\zeta$ between the wall-reference coordinate $x_w$ and the fabrication coordinate $x$, for the upper face sheet; $\zeta$ is a right-handed rotation about $z$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECZU</td>
<td>eccentricity in $z'$ direction, for the upper face sheet; ECZL is the $z'$ coordinate of the shell wall middle surface</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPLASU</td>
<td>upper face sheet material-nonlinearity flag:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>elastic behavior only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>plasticity included, with the material law satisfied at each element integration point</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IANGL</td>
<td>upper face sheet wall-reference option; see discussion on page 8-18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>use default strategy of projecting $x_p, y_g$ to establish $x_w$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>input $r_w$, which is projected to establish $x_w$; see (T-9b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

if ( IANGU > 0 ) then  go to T-9d  
else    go to T-9e
T-9d E830 Upper Face-Sheet Wall Reference Vector

RXU, RYU, RZU upper face sheet wall reference vector, \( \mathbf{r}_w \), which is projected onto the element surface to determine the direction of the wall-reference coordinate \( x_w \); \( \mathbf{r}_w \) is expressed in \( (x_g, y_g, z_g) \) global coordinates

[go to T-9e]
**T-9e E830 Core Properties**

A single T-9e record must be included immediately following the T-9c (or T-9d) record, to specify parameters for the core component of the element(s) specified in the current definition.

**IFABC ZETAC IPLASC IANGC**

**IFABC**
- core fabrication identifier:
  - >0 – wall configuration number in the Wall Fabrication Table (K-1)
  - <0 – solid fabrication identifier in the GCP Fabrication Table (I-22a)

**ZETAC**
- angle $\zeta$ between the wall-reference coordinate $x_w$ and the fabrication coordinate $x$, for the core component; $\zeta$ is a right-handed rotation about $z$

**IPLASC**
- core material-nonlinearity flag:
  - 0 – elastic behavior only
  - 1 – plasticity included (not operational with GCP fabrications, yet)

**IANGC**
- core wall-reference option; see discussion on page 8-18
  - 0 – use default strategy of projecting $x_g, y_g$ to establish $x_w$
  - 1 – input $r_w$, which is projected to establish $x_w$; see (T-9b)

**NX** (T-9)  
- x-direction looping parameter

**NY** (T-9)  
- y-direction looping parameter

- if (IANGC > 0) then go to T-9f
- elseif (NX > 0) then go to T-9g
- elseif (NY > 0) then go to T-9h
- else follow instructions at end of T-9h
**T-9f E830 Core Reference Vector**

**RXC, RYC, RZC** core reference vector, \( r_w \), which is projected onto the element surface to determine the direction of the core-reference coordinate \( x_w \); \( r_w \) is expressed in \( (x_g', y_g', z_g') \) global coordinates

\[
\begin{align*}
\text{if} & \quad (\text{NX} > 0) \quad \text{then} \quad \text{go to T-9g} \\
\text{elseif} & \quad (\text{NY} > 0) \quad \text{then} \quad \text{go to T-9h} \\
\text{else} & \quad \text{follow instructions at end of T-9h}
\end{align*}
\]

NX (T-9) x-direction looping parameter
NY (T-9) y-direction looping parameter
**T-9g E830 X-Direction Incrementations**

A single record of type T-9g must be included immediately after the T-9e or T-9f record when the **NX “x-direction” looping parameter** (T-9) is greater than unity. Eight nodal incrementation variables are specified here for use with the “x-direction” looping function on the T-9 record. **NX E830** sandwich elements are generated in the “x-direction” with this information. Three incrementation variables are also specified here for “x-direction” looping with the user-element-number parameters on T-9.

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>I2</td>
<td>I3</td>
<td>I4</td>
<td>I5</td>
<td>I6</td>
<td>I7</td>
<td>I8</td>
<td>I9</td>
<td></td>
</tr>
</tbody>
</table>

- **I1** incrementation for the **N1** lower face sheet node on the T-9a record
- **I2** incrementation for the **N2** lower face sheet node on the T-9a record
- **I3** incrementation for the **N3** lower face sheet node on the T-9a record
- **I4** incrementation for the **N4** upper face sheet node on the T-9c record
- **I5** incrementation for the **N5** upper face sheet node on the T-9c record
- **I6** incrementation for the **N6** upper face sheet node on the T-9c record
- **I7** incrementation for **USERC** on the T-9 record
- **I8** incrementation for **USER1** on the T-9 record
- **I9** incrementation for **USER2** on the T-9 record

Any of these incrementation variables can be negative, zero, or positive, as required.

**Example:** the following T-9 / T-9a / T-9c / T-9g record combination:

```
830 1 0 0 3 $ T-9 with NX=3
10 20 30 ... $ T-9a lower face nodes
50 60 70 ... $ T-9c upper face nodes
... 1 1 1 2 2 2 $ T-9g increments
```

generates three **E830** elements, with the following nodes:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
<td>30</td>
<td>50</td>
<td>60</td>
<td>70</td>
</tr>
<tr>
<td>11</td>
<td>21</td>
<td>31</td>
<td>52</td>
<td>62</td>
<td>72</td>
</tr>
<tr>
<td>12</td>
<td>22</td>
<td>32</td>
<td>54</td>
<td>64</td>
<td>74</td>
</tr>
</tbody>
</table>

**NY** (T-9) **y-direction looping parameter**

if **NY > 0** then *go to T-9h*
else *follow instructions at end of T-9h*
**T-9h E830 Y-Direction Incrementations**

A single type T-9h record must be included immediately after the T-9e or T-9f or T-9g record when the **NY** “y-direction” looping parameter is greater than unity (T-9). Eight nodal incrementation variables are specified here for use with the T-9 record “y-direction” looping function. **NY E830** sandwich elements are generated in the “y-direction”, for each **E830** element generated in the x-direction, with this information. The **NX** and **NY** looping parameters are usually employed together to specify **NX × NY E830** elements in a single stroke.

<table>
<thead>
<tr>
<th>J1</th>
<th>J2</th>
<th>J3</th>
<th>J4</th>
<th>J5</th>
<th>J6</th>
<th>J7</th>
<th>J8</th>
<th>J9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **J1** incrementation for the **N1** lower face sheet node on the T-9a record
- **J2** incrementation for the **N2** lower face sheet node on the T-9a record
- **J3** incrementation for the **N3** lower face sheet node on the T-9a record
- **J4** incrementation for the **N4** upper face sheet node on the T-9c record
- **J5** incrementation for the **N5** upper face sheet node on the T-9c record
- **J6** incrementation for the **N6** upper face sheet node on the T-9c record
- **J7** incrementation for **USERC** on the T-9 record
- **J8** incrementation for **USER1** on the T-9 record
- **J9** incrementation for **USER2** on the T-9 record

Any of these incrementation variables can be negative, zero, or positive, as required.

**Example:** the following T-9 / T-9a / T-9c / T-9g / T-9h record combination:

```
830 1 0 0 3 2 $ T-9 NX=3, NY=2
10 20 30 ... $ T-9a (lower face nodes)
50 60 70 ... $ T-9c (upper face nodes)
... 1 1 1 1 1 1 $ T-9g (x increments)
5 5 5 5 5 5 $ T-9h (y increments)
```

generates six **E830** elements, with the following nodes:

```
10 20 30 50 60 70  Element #1
11 21 31 51 61 71  Element #2
12 22 32 52 62 72  Element #3
15 25 35 55 65 75  Element #4
16 26 36 56 66 76  Element #5
17 27 37 57 67 77  Element #6
```
N830 (T-5) number of E830 definitions
N840 (T-5) number of E840 definitions
N845 (T-5) number of E845 definitions
N847 (T-5) number of E847 definitions
N849 (T-5) number of E849 definitions
N880 (T-5) number of E880 definitions

if ( fewer than N830 E830 definitions have been processed )return to T-9
elseif ( N840 > 0 ) then go to T-10
elseif ( N845 > 0 ) then go to T-11
elseif ( N847 > 0 ) then go to T-12
elseif ( N849 > 0 ) then go to T-13
elseif ( N880 > 0 ) then go to T-14
else go to U-1
**T-10 E840 8-Node Sandwich Element Definition**

The 8–node **E840** sandwich element in **STAGS** is constructed with a pair of **E410** quadrilateral shells (with bending as well as the usual membrane stiffness) that are held apart by a lightweight core that has three-dimensional elastic properties—as shown in Figure 8.7.

![E840 Sandwich Element](image)

**Figure 8.7** E840 8-Node Sandwich Element

Definition of one or more 8–node **E840** sandwich elements is accomplished with the following set of records:

- **T-10** to specify parameters used for all parts of the element
- **T-10a** to specify information for the lower face sheet of the element
- **T-10b** to specify lower face sheet transformation angles (optional)
- **T-10c** to specify information for the upper face sheet
- **T-10d** to specify upper face sheet transformation angles (optional)
- **T-10e** to specify core parameters
- **T-10f** to specify core transformation angles (optional)
- **T-10g** for “x-direction” incrementation parameters
- **T-10h** for “y-direction” incrementation parameters

These are described sequentially in the following text.
**KELT** ILIN INTEG IPEN NX NY USERC USER1 USER2

**KELT**
- element code number (must be 840)

**ILIN**
- geometric-linearity flag:
  - 0 – nonlinear strain-displacement relations
  - 1 – linear strain-displacement relations

**INTEG**
- integration-type flag (see N-1):
  - 0 – standard integration, or \(2 \times 2\) Gauss points for **E410** elements
  - 1 – modified 5-point integration, previously referred to as full integration

**IPEN**
- penalty option (see N-1)
  - 0 – no penalty function on fourth-order terms in **E410** elements
  - 1 – penalty function included in **E410** elements

**NX**
- x-direction looping parameter, set equal to unity by **STAGS** if omitted or nonpositive; \(\text{NX} > 1\) instructs **STAGS** to generate a set of **E840** elements in the x-direction, using nodal incrementation information given on the T-10g record described below

**NY**
- y-direction looping parameter, set equal to unity by **STAGS** if omitted or nonpositive; \(\text{NY} > 1\) instructs **STAGS** to generate a set of **E840** elements in the y-direction for each of the **NX** elements generated in the x-direction in the current definition, using nodal incrementation information given on the T-10h record described below

**USERC**
- user-specified element number for core component
  (used only if **IUWLE** = 1 on H-1)

**USER1**
- user-specified element number for **lower face sheet**, (if **IUWLE** = 1 on H-1)

**USER2**
- user-specified element number for **upper face sheet**, (if **IUWLE** = 1 on H-1)

\(\rightarrow\) go to T-10a
**T-10a E840 Lower Face-Sheet Properties**

A single T-10a record must be included immediately following the T-10 record, to specify the nodes for the lower face sheet of the first E840 element and to specify other parameters for the lower face sheet of each of the element(s) specified in the current definition.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
<th>IFABL</th>
<th>ZETAL</th>
<th>ECZL</th>
<th>IPLASL</th>
<th>IANGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>first lower face sheet node point</td>
<td>second lower face sheet node point</td>
<td>third lower face sheet node point</td>
<td>fourth lower face sheet node point</td>
<td>fabrication identifier for the lower face sheet element(s); this is equivalent to the WALL parameter for E410 quadrilateral elements:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;0</td>
<td>wall configuration number in the Wall Fabrication Table (K-1)</td>
<td>0</td>
<td>shell wall properties are given in user-written subroutine WALL (See the T-4 record description for more information about this)</td>
<td>&lt;0</td>
<td>shell fabrication identifier in the GCP Fabrication Table (I-21a)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>angle $\zeta$ between the wall-reference coordinate $x_w$ and the fabrication coordinate $x$, for the lower face sheet; $\zeta$ is a right-handed rotation about $z$; see Figure 8.2 on page 8-19.</td>
<td>eccentricity in $z'$ direction, for the lower face sheet; $ECZL$ is the $z'$ coordinate of the shell wall middle surface; please refer to Figure 6.2 on page 6-28; in element units, $(x',y',z')$ are used in place of $(X',Y',Z')$, which do not exist in an element unit; see “Effects of Eccentricity” on page 16-6.</td>
<td>lower face sheet material-nonlinearity flag:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>elastic behavior only</td>
<td>1</td>
<td>plasticity included, with the material law satisfied at each element integration point</td>
<td>2</td>
<td>plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**IANGL**

lower face sheet wall-reference option; see discussion on page 8-18

0  – use default strategy of projecting \(x_g,y_g\) to establish \(x_w\)

1  – input \(r_w\), which is projected to establish \(x_w\); see (T-10b)

\[
\text{if ( IANGL > 0 ) then} \\
\quad \text{go to T-10b} \\
\text{else} \\
\quad \text{go to T-10c}
\]
T-10b E840 Lower Face-Sheet Wall Reference Vector

lower face sheet wall reference vector, \( \mathbf{r}_w \), which is projected onto the element surface to determine the direction of the wall-reference coordinate \( x_w \); \( \mathbf{r}_w \) is expressed in \((x_g, y_g, z_g)\) global coordinates

go to T-10c
**T-10c E840 Upper Face-Sheet Properties**

A single T-10c record must be included immediately following the T-10a (or T-10b) record, to specify the nodes for the upper face sheet of the first E840 element and to specify other parameters for the upper face sheet of each of the element(s) specified in the current definition.

**N5 N6 N7 N8  IFABU  ZETAU  ECZU  IPLASU  IANGU**

- **N5**: first upper face sheet node point
- **N6**: second upper face sheet node point
- **N7**: third upper face sheet node point
- **N8**: fourth upper face sheet node point
- **IFABU**: fabrication identifier for the upper face sheet element(s); this is equivalent to the WALL parameter for E410 quadrilateral elements:
  - >0 – wall configuration number in the Wall Fabrication Table (K-1)
  - 0 – shell wall properties are given in user-written subroutine WALL
  - <0 – shell fabrication identifier in the GCP Fabrication Table (I-21a)
- **ZETAU**: angle $\zeta$ between the wall-reference coordinate $x_w$ and the fabrication coordinate $x$, for the upper face sheet; $\zeta$ is a right-handed rotation about $z$
- **ECZU**: eccentricity in $\varepsilon'$ direction, for the upper face sheet; ECZL is the $\varepsilon'$ coordinate of the shell wall middle surface
- **IPLASU**: upper face sheet material-nonlinearity flag:
  - 0 – elastic behavior only
  - 1 – plasticity included, with the material law satisfied at each element integration point
  - 2 – plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)
- **IANGU**: upper face sheet wall-reference option; see discussion on page 8-18
  - 0 – use default strategy of projecting $x_g, y_g$ to establish $x_w$
  - 1 – input $r_w$, which is projected to establish $x_w$; see (T-10b)

```plaintext
if ( IANGU > 0 ) then  go to T-10d
else  go to T-10e
```
T-10d E840 Upper Face-Sheet Wall Reference Vector

RXU RYU RZU

RXU, RYU, RZU upper face sheet wall reference vector, \( r_w \), which is projected onto the element surface to determine the direction of the wall-reference coordinate \( x_w \); \( r_w \) is expressed in \((x_g, y_g, z_g)\) global coordinates

go to T-10e
**T-10e E840 Core Properties**

A single T-10e record must be included immediately following the T-10c (or T-10d) record, to specify parameters for the core component of the element(s) specified in the current definition.


<table>
<thead>
<tr>
<th>IFABC</th>
<th>core fabrication identifier:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;0</td>
<td>wall configuration number in the <strong>Wall Fabrication Table</strong> (K-1)</td>
</tr>
<tr>
<td>&lt;0</td>
<td>solid fabrication identifier in the <strong>GCP Fabrication Table</strong> (I-22a)</td>
</tr>
</tbody>
</table>

| ZETAC  | angle ζ between the wall-reference coordinate \( x_w \) and the fabrication coordinate \( x \), for the core component; ζ is a right-handed rotation about \( z \) |

<table>
<thead>
<tr>
<th>IPLASC</th>
<th>core material-nonlinearity flag:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>elastic behavior only</td>
</tr>
<tr>
<td>1</td>
<td>plasticity included (not operational with GCP fabrications, yet)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IANGC</th>
<th>core wall-reference option; see discussion on page 8-18</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>use default strategy of projecting ( x_p, y_g ) to establish ( x_w )</td>
</tr>
<tr>
<td>1</td>
<td>input ( r_w ), which is projected to establish ( x_w ); see (T-10b)</td>
</tr>
</tbody>
</table>

\[
x_w \leftarrow (0 < IANGC < 0) \\
x_w \leftarrow (NX > 0) \\
x_w \leftarrow (NY > 0) \\
else (T-10h)
\]

\[
x = (T-10) \\
y = (T-10) \\
follow instructions at end of T-10h
\]
**T-10f E840 Core Reference Vector**

**RXC, RYC, RZC**

RXC, RYC, RZC core reference vector, \( \mathbf{r_w} \), which is projected onto the element surface to determine the direction of the core-reference coordinate \( x_w \); \( \mathbf{r_w} \) is expressed in \((x_g', y_g', z_g')\) global coordinates.

- **NX** (T-10) x-direction looping parameter
- **NY** (T-10) y-direction looping parameter

\[
\begin{align*}
\text{if} \quad ( \textbf{NX} > 0 ) \quad \text{then} & \quad \text{go to } \text{T-10g} \\
\text{elseif} \quad ( \textbf{NY} > 0 ) \quad \text{then} & \quad \text{go to } \text{T-10h} \\
\text{else} \quad return \text{ to } \text{T-10}
\end{align*}
\]
**T-10g E840 X-Direction Incrementations**

A single record of type T-10g must be included immediately after the T-10e or T-10f record when the \( nx \) “x-direction” looping parameter (T-10) is greater than unity. Eight nodal incrementation variables are specified here for use with the “x-direction” looping function on the T-10 record. \( nx \) E840 sandwich elements are generated in the “x-direction” with this information. Three incrementation variables are also specified here for “x-direction” looping with the user-element-number parameters on T-10.

<table>
<thead>
<tr>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>I4</th>
<th>I5</th>
<th>I6</th>
<th>I7</th>
<th>I8</th>
<th>I9</th>
<th>I10</th>
<th>I11</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>incrementation for the ( N1 ) lower face sheet node on the T-10a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2</td>
<td>incrementation for the ( N2 ) lower face sheet node on the T-10a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I3</td>
<td>incrementation for the ( N3 ) lower face sheet node on the T-10a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I4</td>
<td>incrementation for the ( N4 ) lower face sheet node on the T-10a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I5</td>
<td>incrementation for the ( N5 ) upper face sheet node on the T-10c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I6</td>
<td>incrementation for the ( N6 ) upper face sheet node on the T-10c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I7</td>
<td>incrementation for the ( N7 ) upper face sheet node on the T-10c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I8</td>
<td>incrementation for the ( N8 ) upper face sheet node on the T-10c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I9</td>
<td>incrementation for USERC on the T-10 record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I10</td>
<td>incrementation for USER1 on the T-10 record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I11</td>
<td>incrementation for USER2 on the T-10 record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero, or positive, as required.

**Example:** the following T-10 / T-10a / T-10c / T-10g record combination:

```
840 1 0 0 3 $ T-10 with NX=3
10 20 30 40 ... $ T-10a lower face nodes
50 60 70 80 ... $ T-10c upper face nodes
... 1 1 1 1 2 2 2 2 $ T-10g increments
```

generates three E840 elements, with the following nodes:

```
10 20 30 40 50 60 70 80 Element #1
11 21 31 41 52 62 72 82 Element #2
12 22 32 42 54 64 74 84 Element #3
```

\( ny \) (T-10) y-direction looping parameter

if \( \text{ny} > 0 \) then go to T-10h
else follow instructions at end of T-10h
T-10h E840 Y-Direction Incrementations

A single type T-10h record must be included immediately after the T-10e or T-10f or T-10g record when the NY “y-direction” looping parameter is greater than unity (T-10). Eight nodal incrementation variables are specified here for use with the T-10 record “y-direction” looping function. NY E840 sandwich elements are generated in the “y-direction”, for each E840 element generated in the x-direction, with this information. The NX and NY looping parameters are usually employed together to specify NX × NY E840 elements in a single stroke.

J1 J2 J3 J4   J5 J6 J7 J8   J9 J10 J11

J1 incrementation for the N1 lower face sheet node on the T-10a record
J2 incrementation for the N2 lower face sheet node on the T-10a record
J3 incrementation for the N3 lower face sheet node on the T-10a record
J4 incrementation for the N4 lower face sheet node on the T-10a record
J5 incrementation for the N5 upper face sheet node on the T-10c record
J6 incrementation for the N6 upper face sheet node on the T-10c record
J7 incrementation for the N7 upper face sheet node on the T-10c record
J8 incrementation for the N8 upper face sheet node on the T-10c record
J9 incrementation for USERC on the T-10 record
J10 incrementation for USER1 on the T-10 record
J11 incrementation for USER2 on the T-10 record

Any of these incrementation variables can be negative, zero, or positive, as required.

Example: the following T-10 / T-10a / T-10c / T-10g / T-10h record combination:

840 1 0 0 3 2 $ T-10 NX=3, NY=2
10 20 30 40 ... $ T-10a (lower face nodes)
50 60 70 80 ... $ T-10c (upper face nodes)
... 1 1 1 1 1 1 1 1 1 1 1 1 1 1 $ T-10g (x increments)
5 5 5 5 5 5 5 5 $ T-10h (y increments)

generates six E840 elements, with the following nodes:

10 20 30 40 50 60 70 80   Element #1
11 21 31 41 51 61 71 81   Element #2
12 22 32 42 52 62 72 82   Element #3
15 25 35 45 55 65 75 85   Element #4
16 26 36 46 56 66 76 86   Element #5
17 27 37 47 57 67 77 87   Element #6
N840 (T-5) number of E840 definitions
N845 (T-5) number of E845 definitions
N847 (T-5) number of E847 definitions
N849 (T-5) number of E849 definitions
N880 (T-5) number of E880 definitions

if ( fewer than N830 E840 definitions have been processed ) return to T-10
elseif ( N845 > 0 ) then go to T-11
elseif ( N847 > 0 ) then go to T-12
elseif ( N849 > 0 ) then go to T-13
elseif ( N880 > 0 ) then go to T-14
else go to U-1
**T-11 E845 10-Node Sandwich Transition Element Definition**

The 10–node E845 sandwich-transition element, shown in Figure 8.8, is used in shell units to make a transition between one mesh of E840 elements and another mesh of E840 elements that has nodes that are exactly half as far apart as the one in which the E845 element “resides.” It can be used to accomplish the same objective in an element unit. A single T-11 definition may specify one or more E845 sandwich transition elements, as described below.

![E845 10-Node Sandwich Transition Elements](image)

**Figure 8.8 E845 10-Node Sandwich Transition Elements**

Definition of one or more 10–node E845 sandwich transition elements is accomplished with the following set of records:

- **T-11** to specify parameters used for all parts of the element
- **T-11a** to specify information for the lower face sheet of the element
- **T-11b** to specify lower face sheet transformation angles (optional)
- **T-11c** to specify information for the upper face sheet
- **T-11d** to specify upper face sheet transformation angles (optional)
- **T-11e** to specify core parameters
- **T-11f** to specify core transformation angles (optional)
- **T-11g** for “x-direction” incrementation parameters
- **T-11h** for “y-direction” incrementation parameters

These are described sequentially in the following text.
<table>
<thead>
<tr>
<th>KELT</th>
<th>element code number (must be 845)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I LIN</td>
<td>geometric-linearity flag:</td>
</tr>
<tr>
<td></td>
<td>0 – nonlinear strain-displacement relations</td>
</tr>
<tr>
<td></td>
<td>1 – linear strain-displacement relations</td>
</tr>
<tr>
<td>INTEG</td>
<td>integration-type flag (see N-1):</td>
</tr>
<tr>
<td></td>
<td>0 – standard integration, or $2 \times 2$ Gauss points for $E410$ elements</td>
</tr>
<tr>
<td></td>
<td>1 – modified 5-point integration, previously referred to as full integration</td>
</tr>
<tr>
<td>IPEN</td>
<td>penalty option (see N-1)</td>
</tr>
<tr>
<td></td>
<td>0 – no penalty function on fourth-order terms in $E410$ elements</td>
</tr>
<tr>
<td></td>
<td>1 – penalty function included in $E410$ elements</td>
</tr>
<tr>
<td>IEDGE</td>
<td>identifies the $E845$ edge on which node $N5$ is located ($N10$ is above $N5$):</td>
</tr>
<tr>
<td></td>
<td>1 – node $N5$ is between $N3$ and $N4$</td>
</tr>
<tr>
<td></td>
<td>2 – node $N5$ is between $N4$ and $N1$</td>
</tr>
<tr>
<td></td>
<td>3 – node $N5$ is between $N1$ and $N2$</td>
</tr>
<tr>
<td></td>
<td>4 – node $N5$ is between $N2$ and $N3$</td>
</tr>
<tr>
<td>NX</td>
<td>x-direction looping parameter, set equal to unity by STAGS if omitted or nonpositive; $NX \geq 1$ instructs STAGS to generate a set of $NX$ type $E845$ elements in the x direction, using the increments given on the T-11g record described below</td>
</tr>
<tr>
<td>NY</td>
<td>y-direction looping parameter, set equal to unity by STAGS if omitted or nonpositive; $NY \geq 1$ instructs STAGS to generate a set of $NY$ type $E845$ elements in the y direction for each of the $NX$ elements generated in the x direction in the current definition, using the increments given on the T-11h record described below</td>
</tr>
<tr>
<td>USER</td>
<td>user-specified element number for the first $E845$ element generated here (used only if $IUWLE = 1$ on H-1)</td>
</tr>
</tbody>
</table>

*go to* T-11a
T-11a E845 Lower Face-Sheet Properties

A single T-11a record must be included immediately following the T-11 record, to specify the nodes for the lower face sheet of the first E845 element and to specify other parameters for the lower face sheet of each of the element(s) specified in the current definition.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
<th>N5</th>
<th>IFABL</th>
<th>ZETAL</th>
<th>ECZL</th>
<th>IPLASL</th>
<th>IANGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>first lower face sheet node point</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>second lower face sheet node point</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>third lower face sheet node point</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>fourth lower face sheet node point</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>fifth lower face sheet node point</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFABL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>fabrication identifier for the lower face sheet element(s); this is equivalent to the WALL parameter for E410 quadrilateral elements:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>– wall configuration number in the Wall Fabrication Table (K-1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>– shell wall properties are given in user-written subroutine WALL (See the T-4 record description for more information about this)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>– shell fabrication identifier in the GCP Fabrication Table (I-21a) (this option is not operational yet and should not be used)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZETAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>angle $\zeta$ between the wall-reference coordinate $x_w$ and the fabrication coordinate $\bar{x}$, for the lower face sheet; $\zeta$ is a right-handed rotation about $\bar{z}$; see Figure 8.2 on page 8-19.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECZL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>eccentricity in $z'$ direction, for the lower face sheet; ECZL is the $z'$ coordinate of the shell wall middle surface; please refer to Figure 6.2 on page 6-28; in element units, $(x', y', z')$ are used in place of $(X', Y', Z')$, which do not exist in an element unit; see “Effects of Eccentricity” on page 16-6.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPLASL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>lower face sheet material-nonlinearity flag:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>– elastic behavior only</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>– plasticity included, with the material law satisfied at each element integration point</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>– plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**IANGL**

lower face sheet wall-reference option; see discussion on page 8-18

0  – use default strategy of projecting $x_g, y_g$ to establish $x_w$

1  – input $r_w$, which is projected to establish $x_w$; see (T-11b)

```
if ( IANGL > 0 ) then
    go to T-11b
else
    go to T-11c
```
**T-11b E845 Lower Face-Sheet Wall Reference Vector**

\[ \mathbf{RXL}, \mathbf{RYL}, \mathbf{RZL} \]

The lower face sheet wall reference vector, \( \mathbf{r}_w \), which is projected onto the element surface to determine the direction of the wall-reference coordinate \( x_w \); \( \mathbf{r}_w \) is expressed in \( (x_g, y_g, z_g) \) global coordinates.

*go to T-11c*
**T-11c E845 Upper Face-Sheet Properties**

A single T-11c record must be included immediately following the T-11a (or T-11b) record, to specify the nodes for the upper face sheet of the first E845 element and to specify other parameters for the upper face sheet of each of the element(s) specified in the current definition.

\[
\begin{array}{cccccccccc}
\text{N6} & \text{N7} & \text{N8} & \text{N9} & \text{N10} & \text{IFABU} & \text{ZETAU} & \text{ECZU} & \text{IPLASU} & \text{IANGU} \\
\text{first upper face sheet node point} & & & & & & & & & \\
\text{second upper face sheet node point} & & & & & & & & & \\
\text{third upper face sheet node point} & & & & & & & & & \\
\text{fourth upper face sheet node point} & & & & & & & & & \\
\text{fifth upper face sheet node point} & & & & & & & & & \\
\text{fabrication identifier for the upper face sheet element(s); this is equivalent to the} \\
\text{WALL parameter for E410 quadrilateral elements:} \\
>0 & – & \text{wall configuration number in the Wall Fabrication Table (K-1)} & & & & & & & & \\
0 & – & \text{shell wall properties are given in user-written subroutine WALL} & & & & & & & & \\
<0 & – & \text{shell fabrication identifier in the GCP Fabrication Table (I-21a)} & & & & & & & & \\
\text{angle } \zeta & \text{between the wall-reference coordinate } x_w & \text{and the fabrication coordinate} \\
\text{ } & x, & \text{for the upper face sheet; } \zeta & \text{is a right-handed rotation about } z & \text{} & \text{} & \text{} & \text{} & \text{} & \text{} \\
\text{eccentricity in } z' \text{ direction, for the upper face sheet; } \text{ECZL is the } z' \text{ coordinate of} \\
\text{the shell wall middle surface} & & & & & & & & & \\
\text{upper face sheet material-nonlinearity flag:} & & & & & & & & & \\
0 & – & \text{elastic behavior only} & & & & & & & & \\
1 & – & \text{plasticity included, with the material law satisfied at each element} \\
\text{integration point} & & & & & & & & & \\
2 & – & \text{plasticity included, with the material law satisfied at the element} \\
\text{centroid (centroidal plasticity)} & & & & & & & & & \\
\text{upper face sheet wall-reference option; see discussion on page 8-18} & & & & & & & & & \\
0 & – & \text{use default strategy of projecting } x_g, y_g \text{ to establish } x_w & & & & & & & & \\
1 & – & \text{input } r_w, \text{ which is projected to establish } x_w; \text{ see (T-11b)} & & & & & & & & \\
\end{array}
\]

\[\text{if ( IANGU > 0 ) then } \text{ go to T-11d}\]
\[\text{else } \text{ go to T-11e}\]
**T-11d E845 Upper Face-Sheet Wall Reference Vector**

RXU, RYU, RZU

upper face sheet wall reference vector, \( r_w \), which is projected onto the element surface to determine the direction of the wall-reference coordinate \( x_w \); \( r_w \) is expressed in \((x_g, y_g, z_g)\) global coordinates.

*go to T-11e*
**T-11e E845 Core Properties**

A single T-11e record must be included immediately following the T-11c (or T-11d) record, to specify parameters for the core component of the element(s) specified in the current definition.

---

IFABC  ZETAC  IPLASC  IANGC

**IFABC**

core fabrication identifier:

- $> 0$ – wall configuration number in the Wall Fabrication Table (K-1)
- $< 0$ – solid fabrication identifier in the GCP Fabrication Table (I-22a)

**ZETAC**

angle $\zeta$ between the wall-reference coordinate $x_w$ and the fabrication coordinate $x$, for the core component; $\zeta$ is a right-handed rotation about $z$

**IPLASC**

core material-nonlinearity flag:

- 0 – elastic behavior only
- 1 – plasticity included (*not operational with GCP fabrications, yet*)

**IANGC**

core wall-reference option; see discussion on page 8-18

- 0 – use default strategy of projecting $y_g$ to establish $x_w$
- 1 – input $r_w$, which is projected to establish $x_w$; see (T-11b)

**NX** (T-11) x-direction looping parameter

**NY** (T-11) y-direction looping parameter

```
if ( IANGC > 0 ) then go to T-11f
elseif ( NX > 0 ) then go to T-11g
elseif ( NY > 0 ) then go to T-11h
else follow instructions at end of T-10h
```
**T-11f E845 Core Reference Vector**

**RXC  RYC  RZC**

RXC, RYC, RZC core reference vector, \( \mathbf{r}_w \), which is projected onto the element surface to determine the direction of the core-reference coordinate \( x_w \); \( \mathbf{r}_w \) is expressed in \((x',y',z')\) global coordinates.

\[ \begin{align*}
\text{NX} &\quad (T-11) \\
\text{NY} &\quad (T-11)
\end{align*} \]

- **x-direction looping parameter**
- **y-direction looping parameter**

if \((\text{NX} > 0)\) then **go to T-11g**

elseif \((\text{NY} > 0)\) then **go to T-11h**

else follow instructions at end of **T-11h**
**T-11g E845 X-Direction Incrementations**

A single record of type T-11g must be included immediately after the T-11e or T-11f record when the **NX “x-direction” looping parameter** (T-11) is greater than unity. Ten nodal incrementation variables are specified here for use with the “x-direction” looping function on the T-11 record. **NX E845** sandwich transition elements are generated in the “x-direction” with this information. One incrementation variable is also specified here for “x-direction” looping with the user-element-number parameter on T-11.

<table>
<thead>
<tr>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>I4</th>
<th>I5</th>
<th>I6</th>
<th>I7</th>
<th>I8</th>
<th>I9</th>
<th>I10</th>
<th>I11</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>incrementation for the N1 lower face sheet node on the T-11a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2</td>
<td>incrementation for the N2 lower face sheet node on the T-11a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I3</td>
<td>incrementation for the N3 lower face sheet node on the T-11a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I4</td>
<td>incrementation for the N4 lower face sheet node on the T-11a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I5</td>
<td>incrementation for the N5 lower face sheet node on the T-11a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I6</td>
<td>incrementation for the N6 upper face sheet node on the T-11c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I7</td>
<td>incrementation for the N7 upper face sheet node on the T-11c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I8</td>
<td>incrementation for the N8 upper face sheet node on the T-11c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I9</td>
<td>incrementation for the N9 upper face sheet node on the T-11c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I10</td>
<td>incrementation for the N10 upper face sheet node on the T-11c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I11</td>
<td>incrementation for <strong>USER</strong> on the T-11 record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero, or positive, as required.

**Example:** the following T-11 / T-11a / T-11c / T-11g record combination:

```
845 1 0 0 1 3 $ T-11 with NX=3
10 20 30 40 50 ... $ T-11a lower face nodes
60 70 80 90 100 ... $ T-11c upper face nodes
... 1 1 1 1 2 2 2 2 $ T-11g increments
```

generates three **E845** elements, with the following nodes:

```
10 20 30 40 50 60 70 80 90 100 Element #1
11 21 31 41 51 62 72 82 92 102 Element #2
12 22 32 42 52 64 74 84 94 104 Element #3
```

**NY (T-11) y-direction looping parameter**

```
if ( NY > 0 ) then go to T-11h
else follow instructions at end of T-11h
```


**T-11h E845 Y-Direction Incrementations**

A type T-11h record must be included immediately after the T-11e or T-11f or T-11g record when the NY “y-direction” looping parameter is greater than unity (T-11). Ten nodal incrementation variables are specified here for use with the T-11 record “y-direction” looping function. NY E845 sandwich transition elements are generated in the “y-direction”, for each E845 element generated in the x-direction, with this information. The NX and NY looping parameters are usually employed together to specify NX × NY E845 elements in a single stroke.

<table>
<thead>
<tr>
<th>J1</th>
<th>J2</th>
<th>J3</th>
<th>J4</th>
<th>J5</th>
<th>J6</th>
<th>J7</th>
<th>J8</th>
<th>J9</th>
<th>J10</th>
<th>J11</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>incrementation for the N1 lower face sheet node on the T-11a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>incrementation for the N2 lower face sheet node on the T-11a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>incrementation for the N3 lower face sheet node on the T-11a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>incrementation for the N4 lower face sheet node on the T-11a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>incrementation for the N5 lower face sheet node on the T-11a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>incrementation for the N6 upper face sheet node on the T-11c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>incrementation for the N7 upper face sheet node on the T-11c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>incrementation for the N8 upper face sheet node on the T-11c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>incrementation for the N9 upper face sheet node on the T-11c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>incrementation for the N10 upper face sheet node on the T-11c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>incrementation for USER on the T-11 record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero, or positive, as required.

**Example:** the following T-11 / T-11a / T-11c / T-11g / T-11h record combination:

```
845 1 0 0 1 3 2 $ T-11 NX=3, NY=2
10 20 30 40 50 ... $ T-11a (lower face nodes)
60 70 80 90 100 ... $ T-11c (upper face nodes)
...
1 1 1 1 1 1 1 1 1 1 $ T-11g (x increments)
5 5 5 5 5 5 5 5 5 $ T-11h (y increments)
```

generates six E845 elements, with the following nodes:

```
10 20 30 40 50 60 70 80 90 100 Element #1
11 21 31 41 51 61 71 81 91 101 Element #2
12 22 32 42 52 62 72 82 92 102 Element #3
15 25 35 45 55 65 75 85 95 105 Element #4
16 26 36 46 56 66 76 86 96 106 Element #5
17 27 37 47 57 67 77 87 97 107 Element #6
```
N845 (T-5) number of E845 definitions
N847 (T-5) number of E847 definitions
N849 (T-5) number of E849 definitions
N880 (T-5) number of E880 definitions

if ( fewer than N845 E845 definitions have been processed ) return to T-11
elseif ( N847 > 0 ) then go to T-12
elseif ( N849 > 0 ) then go to T-13
elseif ( N880 > 0 ) then go to T-14
else go to U-1
**T-12 E847 14-Node Sandwich Transition Element Definition**

The 14–node **E847** sandwich-transition element, shown in Figure 8.9, is used in shell units to make a transition between one mesh of **E840** elements and two other meshes of **E840** elements that have nodes that are exactly half as far apart as the one in which the **E847** element “resides.” It can be used to accomplish the same objective in an element unit. A single T-12 definition may specify one or more **E847** sandwich transition elements, as described below.

![Figure 8.9 E847 14-Node Sandwich Transition Elements](image)

Definition of one or more 14–node **E847** sandwich transition elements is accomplished with the following set of records:

- **T-12** to specify parameters used for all parts of the element
- **T-12a** to specify information for the lower face sheet of the element
- **T-12b** to specify lower face sheet transformation angles (optional)
- **T-12c** to specify information for the upper face sheet
- **T-12d** to specify upper face sheet transformation angles (optional)
- **T-12e** to specify core parameters
- **T-12f** to specify core transformation angles (optional)
- **T-12g** for “x-direction” incrementation parameters
- **T-12h** for “y-direction” incrementation parameters

These are described sequentially in the following text.
<table>
<thead>
<tr>
<th>KELT</th>
<th>ILIN</th>
<th>INTEG</th>
<th>IPEN</th>
<th>IEDGE</th>
<th>NX</th>
<th>NY</th>
<th>USER</th>
</tr>
</thead>
</table>

**ILIN**
- geometric-linearity flag:
  0 – nonlinear strain-displacement relations
  1 – linear strain-displacement relations

**INTEG**
- integration-type flag (see N-1):
  0 – standard integration, or $2 \times 2$ Gauss points for E410 elements
  1 – modified 5-point integration, previously referred to as full integration

**IPEN**
- penalty option (see N-1)
  0 – no penalty function on fourth-order terms in E410 elements
  1 – penalty function included in E410 elements

**IEDGE**
- identifies the E847 edges on which nodes N5 and N6 are located
  nodes N12 and N13 are above N5 and N6, respectively:
  1 – node N5 is between N3 and N4; node N6 is between N4 and N1
  2 – node N5 is between N4 and N1; node N6 is between N1 and N2
  3 – node N5 is between N1 and N2; node N6 is between N2 and N3
  4 – node N5 is between N2 and N3; node N6 is between N3 and N4

**NX**
- x-direction looping parameter, set equal to unity by STAGS if omitted or nonpositive; $\text{NX} \geq 1$ instructs STAGS to generate a set of NX type E847 elements in the x direction, using the increments given on the T-12g record described below

**NY**
- y-direction looping parameter, set equal to unity by STAGS if omitted or nonpositive; $\text{NY} \geq 1$ instructs STAGS to generate a set of NY type E847 elements in the y direction for each of the NX elements generated in the x-direction in the current definition, using the increments given on the T-12h record described below

**USER**
- user-specified element number for the first E847 element generated here
  (used only if IUWLE = 1 on H-1)

---

Stags 5.0 User Manual  April, 2009  8-81
**T-12a E847 Lower Face-Sheet Properties**

A single T-12a record must be included immediately following the T-12 record, to specify the nodes for the lower face sheet of the first E847 element and to specify other parameters for the lower face sheet of each of the element(s) specified in the current definition.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
<th>N5</th>
<th>N6</th>
<th>N7</th>
<th>IFABL</th>
<th>ZETAL</th>
<th>ECZL</th>
<th>IPLASL</th>
<th>IANGL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **N1**: first lower face sheet node point
- **N2**: second lower face sheet node point
- **N3**: third lower face sheet node point
- **N4**: fourth lower face sheet node point
- **N5**: fifth lower face sheet node point
- **N6**: sixth lower face sheet node point
- **N7**: seventh lower face sheet node point
- **IFABL**: fabrication identifier for the lower face sheet element(s); this is equivalent to the WALL parameter for E410 quadrilateral elements:
  - \( >0 \) – wall configuration number in the Wall Fabrication Table (K-1)
  - \( 0 \) – shell wall properties are given in user-written subroutine WALL (See the T-4 record description for more information about this)
  - \( <0 \) – shell fabrication identifier in the GCP Fabrication Table (I-21a)
- **ZETAL**: angle \( \zeta \) between the wall-reference coordinate \( x_w \) and the fabrication coordinate \( x \), for the lower face sheet; \( \zeta \) is a right-handed rotation about \( z \); see Figure 8.2 on page 8-19.
- **ECZL**: eccentricity in \( z' \) direction, for the lower face sheet; ECZL is the \( z' \) coordinate of the shell wall middle surface; please refer to Figure 6.2 on page 6-28; in element units, \( (x', y', z') \) are used in place of \( (X', Y', Z') \), which do not exist in an element unit; see “Effects of Eccentricity” on page 16-6.
- **IPLASL**: lower face sheet material-nonlinearity flag:
  - \( 0 \) – elastic behavior only
  - \( 1 \) – plasticity included, with the material law satisfied at each element integration point
  - \( 2 \) – plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)
**IANGL**

lower face sheet wall-reference option; see discussion on page 8-18

- 0 – use default strategy of projecting $x_g, y_g$ to establish $x_w$
- 1 – input $r_w$, which is projected to establish $x_w$; see (T-12b)

```plaintext
if ( IANGL > 0 ) then
    go to T-12b
else
    go to T-12c
```
T-12b E847 Lower Face-Sheet Wall Reference Vector

lower face sheet wall reference vector, $r_w$, which is projected onto the
element surface to determine the direction of the wall-reference
coordinate $x_w$; $r_w$ is expressed in $(x_g, y_g, z_g)$ global coordinates

*go to T-12c*
**T-12c E847 Upper Face-Sheet Properties**

A single T-12c record must be included immediately following the T-12a (or T-12b) record, to specify the nodes for the upper face sheet of the first E847 element and to specify other parameters for the upper face sheet of each of the element(s) specified in the current definition.

| N8 | first upper face sheet node point |
| N9 | second upper face sheet node point |
| N10 | third upper face sheet node point |
| N11 | fourth upper face sheet node point |
| N12 | fifth upper face sheet node point |
| N13 | sixth upper face sheet node point |
| N14 | seventh upper face sheet node point |

**IFABU** fabrication identifier for the upper face sheet element(s); this is equivalent to the **WALL** parameter for E410 quadrilateral elements:

- $>0$ – wall configuration number in the **Wall Fabrication Table** (K-1)
- $0$ – shell wall properties are given in user-written subroutine WALL
- $<0$ – shell fabrication identifier in the **GCP Fabrication Table** (I-21a)

**ZETAU** angle $\zeta$ between the wall-reference coordinate $x_w$ and the fabrication coordinate $x$, for the upper face sheet; $\zeta$ is a right-handed rotation about $z$

**ECZU** eccentricity in $z'$ direction, for the upper face sheet; **ECZL** is the $z'$ coordinate of the shell wall middle surface

**IPLASU** upper face sheet material-nonlinearity flag:

- $0$ – elastic behavior only
- $1$ – plasticity included, with the material law satisfied at each element integration point
- $2$ – plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)
IANGL  upper face sheet wall-reference option; see discussion on page 8-18

0  – use default strategy of projecting $x_g,y_g$ to establish $x_w$

1  – input $r_w$, which is projected to establish $x_w$; see (T-12b)

if ( IANGU > 0 ) then  go to T-12d
else  go to T-12e
**T-12d E847 Upper Face-Sheet Wall Reference Vector**

RXU, RYU, RZU upper face sheet wall reference vector, \( \mathbf{r}_w \), which is projected onto the element surface to determine the direction of the wall-reference coordinate \( x_w \); \( \mathbf{r}_w \) is expressed in \((x_g, y_g, z_g)\) global coordinates.

*go to T-12e*
**T-12e E847 Core Properties**

A single T-12e record must be included immediately following the T-12c (or T-12d) record, to specify parameters for the core component of the element(s) specified in the current definition.

---

**IFABC ZETAC IPLASC IANGC**

**IFABC**
- core fabrication identifier:
  - >0 – wall configuration number in the Wall Fabrication Table (K-1)
  - <0 – solid fabrication identifier in the GCP Fabrication Table (I-22a)

**ZETAC**
- angle $\zeta$ between the wall-reference coordinate $x_w$ and the fabrication coordinate $x$, for the core component; $\zeta$ is a right-handed rotation about $z$

**IPLASC**
- core material-nonlinearity flag:
  - 0 – elastic behavior only
  - 1 – plasticity included (*not operational with GCP fabrications, yet*)

**IANGC**
- core wall-reference option; see discussion on page 8-18
  - 0 – use default strategy of projecting $x_g, y_g$ to establish $x_w$
  - 1 – input $r_w$, which is projected to establish $x_w$; see (T-12b)

$\star$

**NX** (T-12)  x-direction looping parameter

**NY** (T-12)  y-direction looping parameter

if (IANGC > 0) then go to T-12f
elseif (NX > 0) then go to T-12g
elseif (NY > 0) then go to T-12h
else follow instructions at end of T-12h
**T-12f E847 Core Reference Vector**

**RXC, RYC, RZC** core reference vector, \( \mathbf{r_w} \), which is projected onto the element surface to determine the direction of the core-reference coordinate \( x_w \); \( \mathbf{r_w} \) is expressed in \((x_g', y_g', z_g')\) global coordinates.

- \( \mathbf{r_w} = (\mathbf{RXC}, \mathbf{RYC}, \mathbf{RZC}) \)

\[
\begin{align*}
\text{if } (\mathbf{NX} > 0) & \quad \text{then go to T-12g} \\
\text{elseif } (\mathbf{NY} > 0) & \quad \text{then go to T-12h} \\
\text{else } & \quad \text{follow instructions at end of T-12h}
\end{align*}
\]
**T-12g E847 X-Direction Incrementations**

A single record of type T-12g must be included immediately after the T-12e or T-12f record when the **nx** “x-direction” **looping parameter** (T-12) is greater than unity. Fourteen nodal incrementation variables are specified here for use with the “x-direction” looping function on the T-12 record. nx E847 sandwich elements are generated in the “x-direction” with this information. One incrementation variable is also specified here for “x-direction” looping with the user-element-number parameter on T-12.

<table>
<thead>
<tr>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>I4</th>
<th>I5</th>
<th>I6</th>
<th>I7</th>
<th>I8</th>
<th>I9</th>
<th>I10</th>
<th>I11</th>
<th>I12</th>
<th>I13</th>
<th>I14</th>
<th>I15</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>incrementation for the N1 lower face sheet node on the T-12a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2</td>
<td>incrementation for the N2 lower face sheet node on the T-12a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I3</td>
<td>incrementation for the N3 lower face sheet node on the T-12a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I4</td>
<td>incrementation for the N4 lower face sheet node on the T-12a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I5</td>
<td>incrementation for the N5 lower face sheet node on the T-12a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I6</td>
<td>incrementation for the N6 lower face sheet node on the T-12a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I7</td>
<td>incrementation for the N7 lower face sheet node on the T-12a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I8</td>
<td>incrementation for the N8 upper face sheet node on the T-12c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I9</td>
<td>incrementation for the N9 upper face sheet node on the T-12c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I10</td>
<td>incrementation for the N10 upper face sheet node on the T-12c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I11</td>
<td>incrementation for the N11 upper face sheet node on the T-12c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I12</td>
<td>incrementation for the N12 upper face sheet node on the T-12c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I13</td>
<td>incrementation for the N13 upper face sheet node on the T-12c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I14</td>
<td>incrementation for the N14 upper face sheet node on the T-12c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I15</td>
<td>incrementation for <strong>USER</strong> on the T-12 record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero, or positive, as required.

NY (T-12) y-direction looping parameter

if ( NY > 0 ) then go to T-12h
else follow instructions at end of T-12h
T-12h E847 Y-Direction Incrementations

A type T-12h record must be included immediately after the T-12e or T-12f or T-12g record when the NY “y-direction” looping parameter is greater than unity (T-12). Fourteen nodal incrementation variables are specified here for use with the T-12 record “y-direction” looping function. NY E847 sandwich elements are generated in the “y-direction”, for each E847 element generated in the x-direction, with this information. The NX and NY looping parameters are usually employed together to specify \(NX \times NY\) E847 elements in a single stroke.

\[
\begin{align*}
\text{J1} & \quad \text{incrementation for the N1 lower face sheet node on the T-12a record} \\
\text{J2} & \quad \text{incrementation for the N2 lower face sheet node on the T-12a record} \\
\text{J3} & \quad \text{incrementation for the N3 lower face sheet node on the T-12a record} \\
\text{J4} & \quad \text{incrementation for the N4 lower face sheet node on the T-12a record} \\
\text{J5} & \quad \text{incrementation for the N5 lower face sheet node on the T-12a record} \\
\text{J6} & \quad \text{incrementation for the N6 lower face sheet node on the T-12a record} \\
\text{J7} & \quad \text{incrementation for the N7 lower face sheet node on the T-12a record} \\
\text{J8} & \quad \text{incrementation for the N8 upper face sheet node on the T-12c record} \\
\text{J9} & \quad \text{incrementation for the N9 upper face sheet node on the T-12c record} \\
\text{J10} & \quad \text{incrementation for the N10 upper face sheet node on the T-12c record} \\
\text{J11} & \quad \text{incrementation for the N11 upper face sheet node on the T-12c record} \\
\text{J12} & \quad \text{incrementation for the N12 upper face sheet node on the T-12c record} \\
\text{J13} & \quad \text{incrementation for the N13 upper face sheet node on the T-12c record} \\
\text{J14} & \quad \text{incrementation for the N14 upper face sheet node on the T-12c record} \\
\text{J15} & \quad \text{incrementation for USER on the T-12 record}
\end{align*}
\]

Any of these incrementation variables can be negative, zero, or positive, as required.

\[
\begin{align*}
\text{N847} & \quad \text{(T-5)} \quad \text{number of E847 definitions} \\
\text{N849} & \quad \text{(T-5)} \quad \text{number of E849 definitions} \\
\text{N880} & \quad \text{(T-5)} \quad \text{number of E880 definitions}
\end{align*}
\]

if \(\text{fewer than N847 E847 definitions have been processed}\) return to T-12
elseif \(\text{N849 > 0}\) then go to T-13
elseif \(\text{N880 > 0}\) then go to T-14
else go to U-1
**T-13 E849 18-Node Sandwich Element Definition**

The 18–node **E849** sandwich element in **STAGS**, shown in Figure 8.10, is constructed with a pair of 9–node **E480** quadrilateral shells (with bending as well as the usual membrane stiffness) that are held apart by a lightweight core that has three-dimensional elastic properties. This type of model is called a sandwich of the second kind. A good example of a problem requiring this model would be one with structural walls made of glass fiber faces and using polyurethane foam as the core material.

![E849 18-Node Sandwich Element Diagram](image)

**Figure 8.10 E849 18-Node Sandwich Element**

Definition of one or more 18–node **E849** sandwich elements is accomplished with the following set of records:

- **T-13** to specify parameters used for all parts of the element
- **T-13a** to specify information for the lower face sheet of the element
- **T-13b** to specify lower face sheet transformation angles (optional)
- **T-13c** to specify information for the upper face sheet
- **T-13d** to specify upper face sheet transformation angles (optional)
- **T-13e** to specify core parameters
- **T-13f** to specify core transformation angles (optional)
- **T-13g** for “x-direction” incrementation parameters
- **T-13h** for “y-direction” incrementation parameters

These are described sequentially in the following text.
KELT  ILIN  INTEG  IPEN  NX  NY  USERC  USER1  USER2

**KELT**
Element code number (must be 849)

**ILIN**
Geometric-linearity flag:

0  –  nonlinear strain-displacement relations
1  –  linear strain-displacement relations

**INTEG**
Integration-type flag: not used for this element; set INTEG = 0

**IPEN**
Penalty option: not used for this element; set IPEN = 0

**NX**
X-direction looping parameter, set equal to unity by STAGS if omitted or nonpositive; \( NX > 1 \) instructs STAGS to generate a set of \( NX \) type E849 elements in the x direction, using nodal incrementation information given on the T-13g record described below

**NY**
Y-direction looping parameter, set equal to unity by STAGS if omitted or nonpositive; \( NY > 1 \) instructs STAGS to generate a set of \( NY \) type E849 elements in the y direction for each of the \( NX \) elements generated in the x-direction in the current definition, using nodal incrementation information given on the T-13h record described below

**USERC**
User-specified element number for core component
(used only if IUWLE = 1 on H-1)

**USER1**
User-specified element number for lower face sheet, (if IUWLE = 1 on H-1)

**USER2**
User-specified element number for upper face sheet, (if IUWLE = 1 on H-1)

*go to T-13a*
**T-13a E849 Lower Face-Sheet Properties**

A single T-13a record must be included immediately following the T-13 record, to specify the nodes for the lower face sheet of the first E849 element (see Figure 8.10 on page 8-92) and to specify other parameters for the lower face sheet of each of the element(s) specified in the current definition.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
<th>N5</th>
<th>N6</th>
<th>N7</th>
<th>N8</th>
<th>N9</th>
<th>IFABL</th>
<th>ZETAL</th>
<th>ECZL</th>
<th>IPLASL</th>
<th>IANGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>node # 1 of the lower face sheet</td>
<td>N2</td>
<td>node # 2 of the lower face sheet</td>
<td>N3</td>
<td>node # 3 of the lower face sheet</td>
<td>N4</td>
<td>node # 4 of the lower face sheet</td>
<td>N5</td>
<td>node # 5 of the lower face sheet</td>
<td>N6</td>
<td>node # 6 of the lower face sheet</td>
<td>N7</td>
<td>node # 7 of the lower face sheet</td>
</tr>
<tr>
<td>IFABL</td>
<td>fabrication identifier for the lower face sheet element(s); this is equivalent to the WALL parameter for E480 quadrilateral elements:</td>
<td>&gt;0 – wall configuration number in the Wall Fabrication Table (K-1)</td>
<td>0 – shell wall properties are given in user-written subroutine WALL</td>
<td>(See the T-4 record description for more information about this)</td>
<td>&lt;0 – shell fabrication identifier in the GCP Fabrication Table (I-21a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZETAL</td>
<td>angle $\zeta$ between the wall-reference coordinate $x_w$ and the fabrication coordinate $\bar{x}$, for the lower face sheet; $\zeta$ is a right-handed rotation about $\zeta$; see Figure 8.2 on page 8-19.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECZL</td>
<td>eccentricity in $\zeta'$ direction, for the lower face sheet; ECZL is the $\zeta'$ coordinate of the shell wall middle surface; please refer to Figure 6.2 on page 6-28; in element units, $(x', y', z')$ are used in place of $(X', Y', Z')$, which do not exist in an element unit; see “Effects of Eccentricity” on page 16-6.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPLASL</td>
<td>lower face sheet material-nonlinearity flag:</td>
<td>0 – elastic behavior only</td>
<td>1 – plasticity included, with the material law satisfied at each element integration point</td>
<td>2 – plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
IANGL lower face sheet wall-reference option; see discussion on page 8-18

0 – use default strategy of projecting $x_g, y_g$ to establish $x_w$

1 – input $r_w$, which is projected to establish $x_w$; see (T-13b)

\[
\begin{align*}
\text{if ( IANGL > 0 ) then} & \quad \text{go to T-13b} \\
\text{else} & \quad \text{go to T-13c}
\end{align*}
\]
**T-13b E849 Lower Face-Sheet Wall Reference Vector**

RXL RYL RZL

lower face sheet wall reference vector, $\mathbf{r}_w$, which is projected onto the element surface to determine the direction of the wall-reference coordinate $x_w$; $\mathbf{r}_w$ is expressed in $(x_g, y_g, z_g)$ global coordinates

*go to* T-13c
**T-13c E849 Upper Face-Sheet Properties**

A single T-13c record must be included immediately following the T-13a (or T-13b) record, to specify the nodes for the upper face sheet of the first E849 element (see Figure 8.10 on page 8-92) and to specify other parameters for the upper face sheet of each of the element(s) specified in the current definition.

<table>
<thead>
<tr>
<th>N10</th>
<th>N11</th>
<th>N12</th>
<th>N13</th>
<th>N14</th>
<th>N15</th>
<th>N16</th>
<th>N17</th>
<th>N18</th>
<th>IFABU</th>
<th>ZETAU</th>
<th>ECZU</th>
<th>IPLASU</th>
<th>IANGU</th>
</tr>
</thead>
<tbody>
<tr>
<td>N10</td>
<td>node # 1 of the upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N11</td>
<td>node # 2 of the upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N12</td>
<td>node # 3 of the upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N13</td>
<td>node # 4 of the upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N14</td>
<td>node # 5 of the upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N15</td>
<td>node # 6 of the upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N16</td>
<td>node # 7 of the upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N17</td>
<td>node # 8 of the upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N18</td>
<td>node # 9 of the upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFABU</td>
<td>fabrication identifier for the upper face sheet element(s); this is equivalent to the WALL parameter for E410 quadrilateral elements:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;0</td>
<td>0 – wall configuration number in the Wall Fabrication Table (K-1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 – shell wall properties are given in user-written subroutine WALL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;0</td>
<td>&lt;0 – shell fabrication identifier in the GCP Fabrication Table (I-21a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZETAU</td>
<td>angle $\zeta$ between the wall-reference coordinate $x_w$ and the fabrication coordinate $x$, for the upper face sheet; $\zeta$ is a right-handed rotation about $\zeta$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECZU</td>
<td>eccentricity in $z'$ direction, for the upper face sheet; ECZL is the $z'$ coordinate of the shell wall middle surface</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPLASU</td>
<td>upper face sheet material-nonlinearity flag:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 – elastic behavior only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1 – plasticity included, with the material law satisfied at each element integration point</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2 – plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IANGU</td>
<td>upper face sheet wall-reference option; see discussion on page 8-18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 – use default strategy of projecting $x_g$ to establish $x_w$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1 – input $r_w$, which is projected to establish $x_w$; see (T-13b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

if ( IANGU > 0 ) then 
  go to T-13d
else
  go to T-13e
**T-13d E849 Upper Face-Sheet Wall Reference Vector**

RXU, RYU, RZU upper face sheet wall reference vector, \( \mathbf{r}_w \), which is projected onto the element surface to determine the direction of the wall-reference coordinate \( x_w \); \( \mathbf{r}_w \) is expressed in \( (x_g, y_g, z_g) \) global coordinates.

*go to T-13e*
**T-13e E849 Core Properties**

A single T-13e record must be included immediately following the T-13c (or T-13d) record, to specify parameters for the core component of the element(s) specified in the current definition.

---

<table>
<thead>
<tr>
<th>IFABC</th>
<th>core fabrication identifier:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;0</td>
<td>wall configuration number in the Wall Fabrication Table (K-1)</td>
</tr>
<tr>
<td>&lt;0</td>
<td>solid fabrication identifier in the GCP Fabrication Table (I-22a)</td>
</tr>
</tbody>
</table>

| ZETAC | angle \( \zeta \) between the wall-reference coordinate \( x_w \) and the fabrication coordinate \( x \), for the core component; \( \zeta \) is a right-handed rotation about \( z \) |

<table>
<thead>
<tr>
<th>IPLASC</th>
<th>core material-nonlinearity flag:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>elastic behavior only</td>
</tr>
<tr>
<td>1</td>
<td>plasticity included <em>(not operational with GCP fabrications, yet)</em></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IANGC</th>
<th>core wall-reference option; see discussion on page 8-18</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>use default strategy of projecting ( x_g, y_g ) to establish ( x_w )</td>
</tr>
<tr>
<td>1</td>
<td>input ( r_w ), which is projected to establish ( x_w ); see (T-13b)</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{NX} & \quad \text{(T-13)} \quad \text{x-direction looping parameter} \\
\text{NY} & \quad \text{(T-13)} \quad \text{y-direction looping parameter} \\
\end{align*}
\]

if \( \text{IANGC} > 0 \) then go to T-13f
elseif \( \text{NX} > 0 \) then go to T-13g
elseif \( \text{NY} > 0 \) then go to T-13h
else follow instructions at end of T-13h
**T-13f E849 Core Reference Vector**

**RXC, RYC, RZC**

RXC, RYC, RZC core reference vector, $\mathbf{r}_w$, which is projected onto the element surface to determine the direction of the core-reference coordinate $x_w$; $\mathbf{r}_w$ is expressed in $(x_g', y_g', z_g)$ global coordinates.

- $\mathbf{nx}$ (T-13) x-direction looping parameter
- $\mathbf{ny}$ (T-13) y-direction looping parameter

if ( $\mathbf{nx} > 0$ ) then go to T-13g
elseif ( $\mathbf{ny} > 0$ ) then go to T-13h
else follow instructions at end of T-13h
**T-13g E849 X-Direction Incrementations**

A single record of type T-13g must be included immediately after the T-13e or T-13f record when the **nx** “x-direction” looping parameter (T-13) is greater than unity. Eighteen nodal incrementation variables are specified here for use with the “x-direction” looping function on the T-13 record. **nx E849** sandwich elements are generated in the “x-direction” with this information. Three incrementation variables are also specified here for “x-direction” looping with the user-element-number parameters on T-13.

\[(IX(k), \ k=1,18) \quad IXC \quad IX1 \quad IX2\]

- **IX(k)**: x-direction incrementation for the kth node, which is on the T-13a record (when \(1 \leq k \leq 9\)), or on the T-13c record (when \(10 \leq k \leq 18\)).
- **IXC**: x-direction incrementation for **USERC** on the T-13 record.
- **IX1**: x-direction incrementation for **USER1** on the T-13 record.
- **IX2**: x-direction incrementation for **USER2** on the T-13 record.

Any of these incrementation variables can be negative, zero, or positive, as required.

**Example:** the following T-13 / T-13a / T-13c / T-13g record combination:

\[
\begin{align*}
849 & \ 1 \ 0 \ 0 \ 3 \quad | \quad \text{$T-13$ with $NX=3$} \\
10 & \ 20 \ 30 \ 40 \ 50 \ 60 \ 70 \ 80 \ 90 \ldots \quad | \quad \text{$T-13a$ lower face nodes} \\
15 & \ 25 \ 35 \ 45 \ 55 \ 65 \ 75 \ 85 \ 95 \ldots \quad | \quad \text{$T-13c$ upper face nodes} \\
\ldots & \\
1 & 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 9*2 \quad | \quad \text{$T-13g$ increments}
\end{align*}
\]

generates three **E849** elements, with the following nodes:

\[
\begin{align*}
10 & \ 20 \ 30 \ 40 \ 50 \ 60 \ 70 \ 80 \ 90 \quad | \quad 15 \ 25 \ 35 \ 45 \ 55 \ 65 \ 75 \ 85 \ 95 \quad \text{Element #1} \\
11 & \ 21 \ 31 \ 41 \ 51 \ 61 \ 71 \ 81 \ 91 \quad | \quad 17 \ 27 \ 37 \ 47 \ 57 \ 67 \ 77 \ 87 \ 97 \quad \text{Element #2} \\
12 & \ 22 \ 32 \ 42 \ 52 \ 62 \ 72 \ 82 \ 92 \quad | \quad 19 \ 29 \ 39 \ 49 \ 59 \ 69 \ 79 \ 89 \ 99 \quad \text{Element #3}
\end{align*}
\]

**NY** (T-13) y-direction looping parameter

- if \((NY > 0)\) then *go to T-13h*
- else *follow instructions at end of T-13h*
**T-13h E849 Y-Direction Incrementations**

A single type T-13h record must be included immediately after the T-13e or T-13f or T-13g record when the NY “y-direction” looping parameter is greater than unity (T-13). Eighteen nodal incrementation variables are specified here for use with the T-13 record “y-direction” looping function. NY E849 sandwich elements are generated in the “y-direction”, for each E849 element generated in the x-direction, with this information. The NX and NY looping parameters are usually employed together to specify NX × NY E849 elements in a single stroke.

\[( IY(k), k=1,18 ) \quad IYC \quad IY1 \quad IY2 \]

- **IY(k)**: y-direction incrementation for the kth node, which is on the T-13a record (when \(1 \leq k \leq 9\)), or on the T-13c record (when \(10 \leq k \leq 18\)).
- **IYC**: y-direction incrementation for USERC on the T-13 record.
- **IY1**: y-direction incrementation for USER1 on the T-13 record.
- **IY2**: y-direction incrementation for USER2 on the T-13 record.

Any of these incrementation variables can be negative, zero, or positive, as required.

\[
\begin{align*}
N_{849} &\quad \text{(T-5) number of E849 definitions} \\
N_{880} &\quad \text{(T-5) number of E880 definitions}
\end{align*}
\]

if \( \text{fewer than } N_{849} \text{ E849 definitions have been processed} \) return to T-13
elseif \( N_{880} > 0 \) then go to T-14
else go to U-1
Solid elements

The current version of STAGS gives the analyst a choice of four solid elements:

- **E881** — an 8-node ANS solid element
- **E882** — an 18-node ANS solid element
- **E883** — a 27-node ANS solid element
- **E885** — a 20-node BR20 displacement-based solid element

The elements and their node-numbering patterns are shown in Figure 8.11:

![Figure 8.11](image-url)
**T-14 E880-Family Solid Element**

The N880 parameter, on T-5, specifies the number of definitions that the analyst wishes to make to include one or more of the E880-family solid elements in the current element unit. Each of these definitions begins with a T-14 record, on which the KELT parameter, indicates which one of these element types is being specified with any given solid-element definition:

<table>
<thead>
<tr>
<th>KELT</th>
<th>IFAB</th>
<th>IANG</th>
<th>ILIN</th>
<th>IPLAS</th>
<th>NX</th>
<th>NY</th>
</tr>
</thead>
<tbody>
<tr>
<td>solid element type:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KELT = 881</td>
<td>8-node ANS solid element</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KELT = 882</td>
<td>18-node ANS solid element</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KELT = 883</td>
<td>27-node ANS solid element</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KELT = 885</td>
<td>20-node BR20 displacement-based solid element</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**IFAB**

fabrication identifier for the element; this is equivalent to the WALL parameter for E410 quadrilateral elements:

- >0 – wall configuration number in the Wall Fabrication Table (K-1)
- 0 – wall properties are given in user-written subroutine WALL
- <0 – fabrication identifier in the GCP Fabrication Table (I-22a)

**IANG**

wall-reference option:

- 0 – use default strategy of projecting x_g, y_g to establish x_w
- 1 – input r_w, which is projected to establish x_w

**IPLAS**

material-nonlinearity flag:

- 0 – elastic behavior only
- 1 – plasticity included, with the material law satisfied at each element integration point
- 2 – plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)

**NX**

number of elements to be generated in the x-direction;

STAGS sets NX = 1 if it is nonpositive

**NY**

number of elements to be generated in the y-direction;

STAGS sets NY = 1 if it is nonpositive

*go to T-14a*
**T-14a E880 Solid Element Nodes**

A single T-14a record must follow the T-14 record, to identify the user points that define the first solid element of the set to be generated here. The number NPTS of user points required is 8 for an E881 element, 18 for an E882 element, 27 for an E883 element, or 20 for an E885 element. These user points must have been defined by appropriate S-1/S-1a/S-2 or S-3/S-3a/S-4 records, as described in Chapter 7.

\[
\text{NODE}(k), \text{k=1,NPTS} \quad \text{USERELT}
\]

NODE\( (k) \) \( k^{th} \) of NPTS node points for the initial E881, E882, E883 or E884 solid element

USERELT user-specified element number, used only if IUWLE = 1 on H-1

The nodal orderings for E880-family elements correspond exactly to the nodal orderings that are used in the PATRAN program.

\[
\begin{align*}
\text{NX} & \quad (T-14) \quad \text{x-direction incrementation flag} \\
\text{NY} & \quad (T-14) \quad \text{y-direction incrementation flag} \\
\text{IANG} & \quad (T-14) \quad \text{wall-reference option} \\
\end{align*}
\]

\[
\begin{align*}
\text{if} \quad (\text{NX} > 1) \quad \text{then go to T-14b} \\
\text{elseif} \quad (\text{NY} > 1) \quad \text{then go to T-14c} \\
\text{elseif} \quad (\text{IANG} > 0) \quad \text{then go to T-14d} \\
\text{else} \quad \text{follow instructions at end of T-14d}
\end{align*}
\]
**T-14b E880 X-Direction Incrementations**

Nine x-direction incrementation variables are specified here for use with the \textit{nx} looping function invoked on the T-14 record and the initial node points established on T-14a.

\begin{itemize}
  \item \textbf{IX1} \hspace{1em} x-direction incrementation variable for node \textit{N1} on T-14a
  \item \textbf{IX2} \hspace{1em} x-direction incrementation variable for node \textit{N2} on T-14a
  \item \textbf{IX3} \hspace{1em} x-direction incrementation variable for node \textit{N3} on T-14a
  \item \textbf{IX4} \hspace{1em} x-direction incrementation variable for node \textit{N4} on T-14a
  \item \textbf{IX5} \hspace{1em} x-direction incrementation variable for node \textit{N5} on T-14a
  \item \textbf{IX6} \hspace{1em} x-direction incrementation variable for node \textit{N6} on T-14a
  \item \textbf{IX7} \hspace{1em} x-direction incrementation variable for node \textit{N7} on T-14a
  \item \textbf{IX8} \hspace{1em} x-direction incrementation variable for node \textit{N8} on T-14a
  \item \textbf{IUX} \hspace{1em} x-direction incrementation variable for use with \textit{USERELT}
\end{itemize}

Any of these incrementation variables can be negative, zero or positive.

\begin{verbatim}
if ( \textbf{NY} > 1) \hspace{1em} then \hspace{1em} go to \textbf{T-14c}
elseif ( \textit{IANG} = 1) \hspace{1em} then \hspace{1em} go to \textbf{T-14d}
else \hspace{1em} follow instructions at end of \textbf{T-14d}
\end{verbatim}
T-14c E880 Y-Direction Incrementations

Nine y-direction incrementation variables are specified here for use with the NY looping function invoked on the T-14 record and the initial node points established on T-880a.

<table>
<thead>
<tr>
<th>IY1</th>
<th>IY2</th>
<th>IY3</th>
<th>IY4</th>
<th>IY5</th>
<th>IY6</th>
<th>IY7</th>
<th>IY8</th>
<th>IUY</th>
</tr>
</thead>
<tbody>
<tr>
<td>y-direction incrementation variable for node N1 on T-14a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y-direction incrementation variable for node N2 on T-14a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y-direction incrementation variable for node N3 on T-14a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y-direction incrementation variable for node N4 on T-14a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y-direction incrementation variable for node N5 on T-14a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y-direction incrementation variable for node N6 on T-14a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y-direction incrementation variable for node N7 on T-14a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y-direction incrementation variable for node N8 on T-14a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y-direction incrementation variable for use with USERELT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero or positive.

```plaintext
if ( IANG = 1 ) then go to T-14d
else follow instructions at end of T-14d
```
**T-14d E880 Material Orientation Record**

If \( \text{IANG} > 0 \) (T-14), a single T-14d record must follow the T-14a record (or T-880b or T-880c record, if applicable) for the current E880 solid element. T-14d specifies the material orientation for each of the E880 elements that are to be generated by the current T-14 element-definition set.

\[
\begin{array}{ccc}
XFX & XFY & XFZ \\
YFX & YFY & YFZ \\
\end{array}
\]

- \( XFX, XFY, XFZ \) vector components establishing the \( x \) orientation of the material
- \( YFX, YFY, YFZ \) vector components establishing the \( x \) orientation of the material

\( \text{N880} \) (T-5) number of E880 definitions

if ( fewer than \( \text{N880} \) E880 definitions have been processed ) return to T-14

else go to U-1
This chapter describes the Ecom (element-command) protocol for specifying some or all of the elements that are to be used in constructing an element unit in a STAGS model. As noted earlier, this protocol is used if and only if the NT1, NT2, NT3, NT4 and NT5 parameters are all set equal to 0 on the H-1 record for the current element unit—or if one or more of the NT1, NT2, NT3 and/or NT4 parameters is positive and the NT5 parameter is 3. The Ecom protocol can be used by itself to define all of the elements in the element unit, or in combination with the (older) Edef protocol to define one or more additional elements after other elements have been defined via T-x and/or T-xx records, as described in Chapter 8.

The Ecom protocol, as its name suggests, is a command-driven procedure (like the GCP facility described in Chapter 5). STAGS responds to command and control information on the following T-100 record, to read and process element-definition input or to stop doing so. The analyst uses a typical command, here, to tell STAGS to process one or more record sets that define one or more spring, beam, triangle, quadrilateral, contact, sandwich, solid or user-defined elements—as described in this chapter. The analyst terminates Ecom-controlled element definitions with a simple ‘END’ command.

After all Edef and Ecom protocol element definitions have been processed, STAGS calls user-written subroutines to generate one or more additional elements (if IUEL is positive on H-1)—and then moves forward to process loading and other input information, as required.
**T-100 Ecom Control or Element-Specification Record**

STAGS examines each T-100 record that it encounters to determine if it begins with a character string (in which case STAGS interprets it as an *Ecom control record*) or if it begins with numeric data (in which case STAGS interprets it as an *Ecom element-specification record*). For reasons that will become clear as this narrative continues, the very first and the very last T-100 records for any given element unit must be Ecom control records. One or more T-100 records of either type can appear in any order between the first and last T-100 records for each element unit.

When the analyst’s current T-100 record is an Ecom control record (starting with a character string called **COMMAND**), STAGS examines the first four “active” characters in **COMMAND**—starting with the first non-blank character and continuing with the next three characters following it—to determine whether or not they uniquely identify one or another of the following thirty case-insensitive Ecom commands that STAGS recognizes:

<table>
<thead>
<tr>
<th>Ecom command</th>
<th>Action instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>E110_Elements</td>
<td>specify one or more additional E110 spring element(s)</td>
</tr>
<tr>
<td>E120_Elements</td>
<td>specify one or more additional E120 rigid link element(s)</td>
</tr>
<tr>
<td>E121_Elements</td>
<td>specify one or more additional E121 soft link element(s)</td>
</tr>
<tr>
<td>E130_Elements</td>
<td>specify one or more additional E130 generalized fastener element(s)</td>
</tr>
<tr>
<td>E210_Elements</td>
<td>specify one or more additional E210 beam element(s)</td>
</tr>
<tr>
<td>E250_Elements</td>
<td>specify one or more additional E250 planar BC element(s)</td>
</tr>
<tr>
<td>E320_Elements</td>
<td>specify one or more additional E320 standard triangle element(s)</td>
</tr>
<tr>
<td>E330_Elements</td>
<td>specify one or more additional E330 MIN3 triangle element(s)</td>
</tr>
<tr>
<td>E410_Elements</td>
<td>specify one or more additional E410 4-node quadrilateral element(s)</td>
</tr>
<tr>
<td>E411_Elements</td>
<td>specify one or more additional E411 quadrilateral element(s)</td>
</tr>
<tr>
<td>E480_Elements</td>
<td>specify one or more additional E480 9-node quadrilateral element(s)</td>
</tr>
<tr>
<td>E510_Elements</td>
<td>specify one or more additional E510 5-node quad mesh transition element(s)</td>
</tr>
<tr>
<td>E710_Elements</td>
<td>specify one or more additional E710 7-node quad mesh transition element(s)</td>
</tr>
<tr>
<td>E810_Elements</td>
<td>specify one or more additional E810 PAD surface/surface contact element(s)</td>
</tr>
<tr>
<td>E820_Elements</td>
<td>specify one or more additional E820 generalized point/surface contact element(s)</td>
</tr>
<tr>
<td>E822_Elements</td>
<td>specify one or more additional E822 line/line contact element(s)</td>
</tr>
<tr>
<td>E830_Elements</td>
<td>specify one or more additional E830 6-node sandwich element(s)</td>
</tr>
<tr>
<td>E840_Elements</td>
<td>specify one or more additional E840 8-node sandwich element(s)</td>
</tr>
<tr>
<td>E845_Elements</td>
<td>specify one or more additional E845 10-node sandwich transition element(s)</td>
</tr>
<tr>
<td>E847_Elements</td>
<td>specify one or more additional E847 14-node sandwich transition element(s)</td>
</tr>
<tr>
<td>E849_Elements</td>
<td>specify one or more additional E849 18-node sandwich element(s)</td>
</tr>
<tr>
<td>E881_Elements</td>
<td>specify one or more additional E881 8-node ANS brick element(s)</td>
</tr>
<tr>
<td>E882_Elements</td>
<td>specify one or more additional E882 18-node ANS solid element(s)</td>
</tr>
<tr>
<td>E883_Elements</td>
<td>specify one or more additional E883 27-node ANS solid element(s)</td>
</tr>
<tr>
<td>E885_Elements</td>
<td>specify one or more additional E885 20-node displacement-based solid element(s)</td>
</tr>
<tr>
<td>E900_Elements</td>
<td>specify one or more additional E900 user-defined element(s)</td>
</tr>
<tr>
<td>E928_Elements</td>
<td>specify one or more additional E928 “built-in” 3-node curved-beam UEL(s)</td>
</tr>
<tr>
<td>E940_Elements</td>
<td>specify one or more additional E940 “built-in” MIN4 quadrilateral UEL(s)</td>
</tr>
<tr>
<td>E9XX_Elements</td>
<td>specify one or more additional E9xx UEL(s)</td>
</tr>
</tbody>
</table>

and

**End** — stop specifying elements!
STAGS ignores the ‘_’ character in the user’s COMMAND string and the program’s matrix of recognizable keywords, and everything after that character in both places, so the user can abbreviate any of these keywords to four characters as and if he wishes to do so—except for the 3-character ‘End’ keyword, which he cannot abbreviate.

When the “active” part of the COMMAND string on the user’s current T-100 record “matches” the first four characters of one of the twenty nine ‘Ennn_Elements’ Ecom commands that STAGS recognizes and supports, STAGS understands that the analyst wants to add one or more type ‘nnn’ elements to the current element unit. Under these circumstances, the ‘nnn’ part of the user’s COMMAND must be one or another of the twenty nine 3-digit element-type codes that the current version of the STAGS program supports—one or another of the KELT values that are specified in the nine element-type “groups” that are shown in the following Table:

<table>
<thead>
<tr>
<th>Group</th>
<th>Element-type code (KELT = nnn value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>110 120 121 130</td>
</tr>
<tr>
<td>200</td>
<td>210 250</td>
</tr>
<tr>
<td>300</td>
<td>320 330</td>
</tr>
<tr>
<td>400</td>
<td>410 411 480 510 710</td>
</tr>
<tr>
<td>800</td>
<td>810 820 822</td>
</tr>
<tr>
<td>830</td>
<td>830</td>
</tr>
<tr>
<td>840</td>
<td>840 845 847 849</td>
</tr>
<tr>
<td>880</td>
<td>881 882 883 885</td>
</tr>
<tr>
<td>900</td>
<td>900 928 940 9xx</td>
</tr>
</tbody>
</table>

These twenty nine element-type codes are arranged in “groups” like this because element-specification input requirements tend to be similar from one type of element to another when the elements are in the same group, and because they tend to be very different from each other when the elements are in different element-type groups. It takes more input (of different kinds) for the user to specify an E410 quadrilateral element, for example, than it does for him to specify any type of “spring” element. It also takes more input to specify a set of 18-node E885 elements than it does to specify a set of E881s.

The “real” reason for arranging these element-type codes in nine “groups” like this is to make it easier for the user to specify the elements that he wants to add to his current element unit—by relaxing the old rule that he had to warn STAGS each time he stopped specifying elements of one type and started specifying elements of a different type (with a T-100 record specifying the new type and the number of record sets needed to define those elements). The current version of STAGS lets the user switch from one type of element to any other type of element in the same element-type group—without using an Ecom record to specify the new type and without counting the number of record sets to be used to define them.
With the previous version of STAGS, the Ecom-controlled part of the user’s element-specification input stream might have looked like this (for example):

```
: E220_Elements
   _T-220 record set # 1 (T-220/T-220a)
E221_Elements
   _T-221 record set # 1 (T-221/T-221a)
E410_Elements
   T-410 record set # 1 (T-410/T-410a/T-410b/T-410c)
   T-410 record set # 2 (T-410/T-410a/T-410b/T-410c)
E510_Elements
   _T-510 record set # 1 (T-510/T-510a/T-510b/T-510c)
E710_Elements
   _T-710 record set # 1 (T-710/T-710a/T-710b/T-710c)
E410_Elements
   T-410 record set # 3 (T-410/T-410a/T-410b/T-410c)
   T-410 record set # 4 (T-410/T-410a/T-410b/T-410c)
End
```

With the current version of STAGS, this user might want to take advantage of this new flexibility by removing the “optional” ‘Ennn_Elements’ commands in his input stream and restraining himself from counting the number of element-specification record sets that he needs to do the job at hand—thusly:

```
: E410_Elements
   _T-410 record set # 1 (T-410/T-410a/T-410b/T-410c)
   T-410 record set # 2 (T-410/T-410a/T-410b/T-410c)
   T-410 record set # 3 (T-410/T-410a/T-410b/T-410c)
   T-410 record set # 4 (T-410/T-410a/T-410b/T-410c)
   _T-510 record set # 1 (T-510/T-510a/T-510b/T-510c)
   _T-710 record set # 1 (T-710/T-710a/T-710b/T-710c)
E120_Elements
   _T-220 record set # 1 (T-220/T-220a)
   _T-221 record set # 1 (T-221/T-221a)
End
```

In any event, the reader should take pains to note that the first record in both of these element-specification streams must be an Ecom command that identifies the first type of element that the user wants to add to the current element unit—and that the last Ecom record in both of these element-specification streams must be the ‘END’ command, to stop specifying more elements for that unit. It’s no big deal for the user to cut out four element-type Ecom commands in this small and simple example, but it might make life much simpler and easier for him to do so (without having to count his record sets) for larger and/or more complex cases.

This is a good point for us to remind the reader (again) that when the character string on the analyst’s T-100 (Ecom) record gives STAGS the ‘END’ instruction, the program stops reading element specifications for the current unit and starts reading the user’s U-1 (Loads Summary) input.
This is also a good point for us to tell (or remind) the reader that when the current T-100 record starts with numeric data, STAGS considers that T-100 record to be the first record in the record set that is needed to specify type-nnn elements. Initially (and typically), nnn is the KELT-specification part of the user’s most recent Ecom control command. It will be different from that, however, if the user has switched to a different type of element in the same element-type group. STAGS expects to continue reading and processing one T-nnn record set after another—as described below and in the remainder of this chapter—until the user gives it a new Ecom control record that directs it to do otherwise.

The following “overview” navigation diagram describes that more succinctly than more paragraphs in this “Model Input—Element Units” chapter can:

```
if ( the T-100 record contains an Ecom control command ) then
  if ( COMMAND := ‘Ennn’ --> valid STAGS element type ) then
    go to T-nnn to specify one or more type nnn elements to be
    added to the current element unit
  elseif ( COMMAND := ‘End’ ) then
    go to U-1 (stop specifying elements)
  else
    STOP gracefully (unrecognized command)
  endif
endif
elseif ( the T-100 record contains numerical data ) then
  go to T-nnn and follow instructions there
endif
```

The following navigation diagram describes more precisely what STAGS expects the user to “do” when he reaches (or returns to) this point in preparing his case.inp input file for the s1 program:
if ( T-100 contains an Ecom control command ) then
  if ( COMMAND := 'Ennn' --> valid STAGS element type ) then
    if ( COMMAND = 'E110_Elements' ) then go to T-110
    elseif ( COMMAND = 'E120_Elements' ) then go to T-120
    elseif ( COMMAND = 'E121_Elements' ) then go to T-121
    elseif ( COMMAND = 'E130_Elements' ) then go to T-130
    elseif ( COMMAND = 'E210_Elements' ) then go to T-210
    elseif ( COMMAND = 'E250_Elements' ) then go to T-250
    elseif ( COMMAND = 'E320_Elements' ) then go to T-320
    elseif ( COMMAND = 'E330_Elements' ) then go to T-330
    elseif ( COMMAND = 'E410_Elements' ) then go to T-410
    elseif ( COMMAND = 'E411_Elements' ) then go to T-411
    elseif ( COMMAND = 'E480_Elements' ) then go to T-480
    elseif ( COMMAND = 'E510_Elements' ) then go to T-510
    elseif ( COMMAND = 'E710_Elements' ) then go to T-710
    elseif ( COMMAND = 'E810_Elements' ) then go to T-810
    elseif ( COMMAND = 'E820_Elements' ) then go to T-820
    elseif ( COMMAND = 'E822_Elements' ) then go to T-822
    elseif ( COMMAND = 'E830_Elements' ) then go to T-830
    elseif ( COMMAND = 'E840_Elements' ) then go to T-840
    elseif ( COMMAND = 'E845_Elements' ) then go to T-845
    elseif ( COMMAND = 'E847_Elements' ) then go to T-847
    elseif ( COMMAND = 'E849_Elements' ) then go to T-849
    elseif ( COMMAND = 'E881_Elements' ) then go to T-881
    elseif ( COMMAND = 'E882_Elements' ) then go to T-882
    elseif ( COMMAND = 'E883_Elements' ) then go to T-883
    elseif ( COMMAND = 'E885_Elements' ) then go to T-885
    elseif ( COMMAND = 'E900_Elements' ) then go to T-900
    elseif ( COMMAND = 'E928_Elements' ) then go to T-928
    elseif ( COMMAND = 'E940_Elements' ) then go to T-940
    elseif ( COMMAND = 'E9XX_Elements' ) then go to T-900
  elseif ( COMMAND := 'End' ) then
    go to U-1 (stop specifying elements)
  endif
elseif ( T-100 contains numerical data ) then
  go to T-100
endif
9.1 Definition of “Spring” Elements via the Ecom Protocol

There are four so-called “spring” elements available in the current version of the STAGS program.

The 2-node E110 Mount element is a special-purpose function (disguised as an element) that is designed to model a user-specified displacement-velocity-force profile. One or more E110 elements (or additional E110 elements) can be defined with the T-110 record set. For more information about the E110 element, see “I-4a Mount Element Table Size” on page 5-64 and “E110 Mount element” on page 14-6.

The 2-node E120 rigid link element constrains the distance between two nodes to be invariant during an analysis and the rotations at the two nodes to be equal. It is a traditional rigid link. One or more E120 elements (or additional E120 elements) can be defined with the T-120 record set. For more information about the E120 element, see “E120 Rigid link element” on page 14-7.

The 3-node E121 soft link element constrains three nodes to be colinear. The E121 element is most useful in connecting shell units that are modeled with three-dimensional solid elements to other shell units that are modeled with two-dimensional shell elements. One or more E121 elements (or additional E121 elements) can be defined with the T-121 record set. For more information about the E121 element, see “E121 Soft link element” on page 14-7 and the STAGS Elements Manual.

The 2-node E130 element is a special-purpose element that uses Mount Element Table and other user-supplied information to model breakable fasteners. One or more E130 elements (or additional E130 elements) can be defined with the T-130 record set. For more information about the E130 element, see “E130 Generalized fastener element” on page 14-8.
**T-110 Additional E110 Elements**

This is the first record of the two-record T-110/T-110a Ecom protocol counterpart of the T-1/T-1a record set that also defines one or more E110 mount elements. With this T-110 record (and its T-110a companion), the user can specify one or more additional mount elements to be included in the current element unit. The *looping* capability provided here *via* **NX**, **INC1**, **INC2** and **INC3** can be used effectively in many situations.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>KELT</th>
<th>NX</th>
<th>INC1</th>
<th>INC2</th>
<th>INC3</th>
<th>USERELT</th>
<th>INC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>node 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>node 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>node 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KELT</td>
<td>must = 110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NX</td>
<td>number of mount elements to be specified; <strong>STAGS</strong> sets <strong>NX</strong> = 1 if it is nonpositive or omitted</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC1</td>
<td>incrementation variable for <strong>N1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC2</td>
<td>incrementation variable for <strong>N2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC3</td>
<td>incrementation variable for <strong>N3</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USERELT</td>
<td>user-specified element number (only used if <strong>IUWLE</strong>=1 on <strong>H-1</strong>)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC4</td>
<td>incrementation variable for <strong>USERELT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*go to* T-110a
**T-110a E110 Element Data**

The mount element is described fully in “E110 Mount element” on page 14-6 and in Chapter 8, in connection with the T-1/T-1a record set. As noted there, it is a special nonlinear spring capable of modeling a user-defined displacement-velocity-force profile. Rigid links with this element provide a method for defining rotational stiffness in addition to axial spring stiffness.

<table>
<thead>
<tr>
<th>IMNT1</th>
<th>IMNT2</th>
<th>RLX1</th>
<th>RLY1</th>
<th>RLZ1</th>
<th>RLX2</th>
<th>RLY2</th>
<th>RLZ2</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMNT1</td>
<td>required Mount Element Table identifier (l-4a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMNT2</td>
<td>optional additional mount table identifier: if IMNT2 is not zero, then IMNT1 and IMNT2 must refer to tables consisting of a single row (displacement table, NRV = 1, l-4a) or column (velocity table, NRD = 1, l-4a). There must be one table of each type, or an error will result. If IMNT2 &gt; 0, the total mount force is the sum of the force from the displacement table and the force from the velocity table.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLX1, RLY1, RLZ1</td>
<td>(X₁, Y₁, Z₁) coordinates of point RL1; see Figure 14.1 on page 14-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLX2, RLY2, RLZ2</td>
<td>(X₂, Y₂, Z₂) coordinates of point RL2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

if ( the user wants to add another set of E110 elements ) then
   go to T-110
elseif ( the user wants to add a set of E120 elements ) then
   go to T-120
elseif ( the user wants to add a set of E121 elements ) then
   go to T-121
elseif ( the user wants to add a set of E130 elements ) then
   go to T-130
else
   return to T-100
endif
**T-120 Additional E120 Elements**

This is the first record of the two-record T-120/T-120a Ecom protocol counterpart of the T-1/T-1b record set that also defines one or more E120 rigid link elements. With this record (and T-120a), the user can specify one or more additional rigid link elements to be included in the current element unit. The looping capability provided here via NX, INC1, INC2 and INC3 can be used effectively in many situations.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>KELT</th>
<th>NX</th>
<th>INC1</th>
<th>INC2</th>
<th>INC3</th>
<th>USERELT</th>
<th>INC4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **N1** node 1
- **N2** node 2
- **N3** node 3 — a reference node; a positive value enforces rotational compatibility between N1 and N2
- **KELT** must = 120
- **NX** number of rigid link elements to be specified; STAGS sets NX = 1 if it is nonpositive or omitted
- **INC1** incrementation variable for N1
- **INC2** incrementation variable for N2
- **INC3** incrementation variable for N3
- **USERELT** user-specified element number (only used if IUWLE=1 on H-1)
- **INC4** incrementation variable for USERELT

The third node, N3, is a reference node that is used to enforce rotational compatibility between N1 and N2. If N3 is zero, the freedoms on N2 are independent of those on N1—a condition that simulates a ball joint at N2. If N3 is nonzero, total rotational compatibility is enforced between N1 and N2—a condition that simulates the ordinary rigid link. If N3 is positive, the rotational constraint is enforced with a set of three Lagrange multipliers. For this option, computational coordinates for each node in the link are completely independent (at the discretion of the user), affording maximum flexibility. If N3 is negative, the rotational constraint is enforced using partial compatibility (see G-2, with ID1 = -2). A restriction is introduced that forces the computational coordinates of N2 to be the same as for the master node to which N1 refers. This can have unforeseen consequences if rigid links cause new master/slave relationships to be defined between other nodes in the model. It is recommended that a negative value for N3 be used with caution.

*go to T-120a*
**T-120a E120 Element Data**

The rigid link element is discussed in the *STAGS Elements Manual*. A rigid link element constrains the distance between two nodes (\textbf{N1} and \textbf{N2}) to be invariant during an analysis. The displacements of node \textbf{N2} are dependent on the displacements and rotations of node \textbf{N1} through a rigid-link constraint equation, which is enforced *via* Lagrange multipliers.

The potential energy $\Pi$ for a rigid link element is given by

$$\Pi = \phi \alpha^T (x_1 + r - x_2)$$

where $\alpha$ is a vector of three Lagrange multipliers, $x_1$ and $x_2$ are the current global positions of nodes \textbf{N1} and \textbf{N2}, respectively, and $r$ is the directed position of \textbf{N2} with respect to \textbf{N1}. The magnitude of $r$ is the original distance between nodes \textbf{N1} and \textbf{N2}. The user-specified element scale factor $\phi$ is employed to make the magnitude of the stiffness contributions of the rigid link element comparable in size to those of other elements in the configuration—this value is typically on the order of the elastic modulus of the material.

Note that the rotations on \textbf{N2} are unaffected unless the reference node \textbf{N3} is nonzero, in which case G-2 records are automatically generated to constrain the rotations on \textbf{N1} to be the same as on \textbf{N2}; separate G-2 records can always be generated to constrain the rotational freedoms as the user may desire.

---

**SCALE**

**SCALE** element scale factor

```plaintext
if ( the user wants to add another set of E120 elements ) then
go to T-120
elseif ( the user wants to add a set of E110 elements ) then
go to T-110
elseif ( the user wants to add a set of E121 elements ) then
go to T-121
elseif ( the user wants to add a set of E130 elements ) then
go to T-130
else
   return to T-100
endif
```
**T-121 Additional E121 Elements**

The soft link element, which is described fully in the *STAGS Elements Manual*, uses Lagrange multipliers to constrain the three nodes that are associated with it to be colinear.

T-121 is the first record of the two-record T-121/T-121a Ecom protocol counterpart of the T-1/T-1b record set that also defines one or more E121 soft link elements. With this record (and T-121a), the user can specify one or more additional soft link elements to be included in the current element unit. The *looping* capability provided here *via* NX, INC1, INC2 and INC3 can be used effectively in many situations.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>KELT</th>
<th>NX</th>
<th>INC1</th>
<th>INC2</th>
<th>INC3</th>
<th>USERELT</th>
<th>INC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>active soft-link node # 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>active soft-link node # 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>active soft-link node # 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KELT</td>
<td>must = 121</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NX</td>
<td>number of soft link elements to be specified; STAGS sets NX = 1 if it is nonpositive or omitted</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC1</td>
<td>incrementation variable for N1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC2</td>
<td>incrementation variable for N2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC3</td>
<td>incrementation variable for N3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USERELT</td>
<td>user-specified element number (only used if IUWLE=1 on H-1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC4</td>
<td>incrementation variable for USERELT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*go to* T-121a
**T-121a E121 Element Data**

The basic scale factor for the Lagrange multipliers generated by the soft link element(s) defined here is specified on this record.

| SCALE | basic Lagrange multiplier scale factor |

if ( the user wants to add another set of **E121** elements ) then

   go to **T-121**

elseif ( the user wants to add a set of **E110** elements ) then

   go to **T-110**

elseif ( the user wants to add a set of **E120** elements ) then

   go to **T-120**

elseif ( the user wants to add a set of **E130** elements ) then

   go to **T-130**

else

   return to **T-100**

endif
**T-130 Additional E130 Elements**

The generalized fastener element is described fully in the *STAGS Elements Manual*.

T-130 is the first record of the two-record T-130/T-130a Ecom protocol counterpart of the T-1/ T-1c record set that also defines one or more E130 generalized fastener elements. With this record (and T-130a), the user can specify one or more additional generalized fastener elements to be included in the current element unit. The *looping* capability provided here *via* NX, INC1, INC2 and INC3 can be used effectively in many situations.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>KELT</th>
<th>NX</th>
<th>INC1</th>
<th>INC2</th>
<th>INC3</th>
<th>USERELT</th>
<th>INC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>active soft-link node # 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>active soft-link node # 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>active soft-link node # 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KELT</td>
<td>must = 130</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NX</td>
<td>number of soft link elements to be specified; STAGS sets NX = 1 if it is nonpositive or omitted</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC1</td>
<td>incrementation variable for N1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC2</td>
<td>incrementation variable for N2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC3</td>
<td>incrementation variable for N3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USERELT</td>
<td>user-specified element number (only used if IUWLE=1 on H-1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC4</td>
<td>incrementation variable for USERELT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*go to* T-130a
**T-130a E130 Element Data**

The T-130a record identifies all of the Mount tables and breakage codes that are to be used for *all* of the **E130** mount element(s) that are defined on the current T-130 record.

<table>
<thead>
<tr>
<th>IMNT1</th>
<th>Mount Table identifier (see I–4a) applied to the relative local x translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMNT2</td>
<td>Mount Table identifier (see I–4a) applied to the relative local y translation</td>
</tr>
<tr>
<td>IMNT3</td>
<td>Mount Table identifier (see I–4a) applied to the relative local z translation</td>
</tr>
<tr>
<td>IMNT4</td>
<td>Mount Table identifier (see I–4a) applied to the relative local $\theta_x$ rotation</td>
</tr>
<tr>
<td>IMNT5</td>
<td>Mount Table identifier (see I–4a) applied to the relative local $\theta_y$ rotation</td>
</tr>
<tr>
<td>IMNT6</td>
<td>Mount Table identifier (see I–4a) applied to the relative local $\theta_z$ rotation</td>
</tr>
<tr>
<td>PLAS1</td>
<td>breakage code for <strong>IMNT1</strong>: see discussion, below, for significance</td>
</tr>
<tr>
<td>PLAS2</td>
<td>breakage code for <strong>IMNT2</strong>: see discussion, below, for significance</td>
</tr>
<tr>
<td>PLAS3</td>
<td>breakage code for <strong>IMNT3</strong>: see discussion, below, for significance</td>
</tr>
<tr>
<td>PLAS4</td>
<td>breakage code for <strong>IMNT4</strong>: see discussion, below, for significance</td>
</tr>
<tr>
<td>PLAS5</td>
<td>breakage code for <strong>IMNT5</strong>: see discussion, below, for significance</td>
</tr>
<tr>
<td>PLAS6</td>
<td>breakage code for <strong>IMNT6</strong>: see discussion, below, for significance</td>
</tr>
</tbody>
</table>

The six breakage codes specified here control generalized fastener breakage modes for the associated mount identifiers. Each breakage code may have one of the following five values:

- **PLAS**$^j$ = 1 $\Rightarrow$ fastener $^j$ behaves elastically; its failure *does not* cause other fasteners to fail
- **PLAS**$^j$ = 2 $\Rightarrow$ fastener $^j$ behaves elastically; its failure causes *all* of the fasteners to fail
- **PLAS**$^j$ = 3 $\Rightarrow$ fastener $^j$ behaves elastically, and is unbreakable, throughout the analysis
- **PLAS**$^j$ = 4 $\Rightarrow$ fastener $^j$ behaves plastically; its failure *does not* cause other fasteners to fail
- **PLAS**$^j$ = 5 $\Rightarrow$ fastener $^j$ behaves plastically; its failure causes *all* of the fasteners to fail

```plaintext
if ( the user wants to add another set of **E130** elements ) then
go to T-130
elseif ( the user wants to add a set of **E110** elements ) then
go to T-110
elseif ( the user wants to add a set of **E120** elements ) then
go to T-120
elseif ( the user wants to add a set of **E121** elements ) then
go to T-121
else
  return to T-100
endif
```
9.2 Definition of “Beam” Elements via the Ecom Protocol

There are two standard “beam” elements in the current version of STAGS.

The standard 2-node E210 element is a multi-sectional straight beam that was implemented in the earliest versions of STAGS several decades ago using traditional element-implementation methodologies. One or more E210 elements (or additional E210 elements) can be defined with the T-210 record set—as described on the following two pages. For more information about the E210 element, please see “E210 Beam element” on page 14-13.

The standard 3-node E250 element is a special-purpose function (that is disguised as a “beam element”) that can be used (in groups) to enforce planar boundary conditions. This “element” was implemented in STAGS several lustre ago using traditional methodologies. One or more E250 elements (or additional E250 elements) can be defined with the T-250 record set—as described later in this section. For more information about the E250 element, see the STAGS Elements Manual.

There is one user-developed “beam” element in the current version of STAGS.

The 3-node E928 curved beam element was originally implemented as a User-Defined STAGS element using the STAGS UEL methodologies and standards that are described in Section 9.8 of this chapter and in Chapter 13 of this document. In 2008, this E928 element was integrated into the current version of STAGS as a “built-in” UEL that can be used by anyone who needs a curved beam element that is compatible with STAGS’ standard 9-node E480 quad element. One or more E928 elements can be defined with the T-900 record and the T-928 record set that are also described in Section 9.8 of this chapter. For more information about the E928 element, please see Chapter 14 in this document, the STAGS Elements Manual, and the STAGS Test Cases Manual.
**T-210 Additional E210 Elements**

T-210 is the first record of the two-record T-210/T-210a Ecom protocol counterpart of the T-2/T-2a record set that also defines one or more E210 beam elements. With this record (and T-210a), the user can specify one or more additional generalized fastener elements to be included in the current element unit. The looping capability provided here, via **NX** and the four incrementation parameters on T-210a, can be used effectively in many situations.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>KELT</th>
<th>ICROSS</th>
<th>XSI</th>
<th>ECY</th>
<th>ECZ</th>
<th>ILIN</th>
<th>IPLAS</th>
<th>NX</th>
<th>USERELT</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>active beam node # 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>active beam node # 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>node # 3 — a reference node used to generate the element coordinate system</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KELT</td>
<td>must = 210</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICROSS</td>
<td>cross-section identifier, in the Cross Section Table: see the T-2 description, in Chapter 8, for important information about ICROSS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XSI</td>
<td>angle, in degrees, between the element normal and the cross section. is a right-handed rotation about , the longitudinal axis of the beam, which is parallel to .</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECY</td>
<td>eccentricity in the direction. is the coordinate of the pair which positions the origin of the system.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECZ</td>
<td>eccentricity in the direction. is the coordinate of the pair which positions the origin of the system.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ILIN</td>
<td>geometric nonlinearity flag</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 — nonlinear strain-displacement relations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 — linear strain-displacement relations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPLAS</td>
<td>material nonlinearity flag (see M-5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 — linear elastic constitutive relations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 — plasticity included</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 — centroidal plasticity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NX</td>
<td>number of beam elements to be generated via this T-210 record; set to unity by STAGS if nonpositive or omitted</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USERELT</td>
<td>user-specified element number, used only if IUWLE = 1 on H-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

if (NX > 1) then  
go to T-210a  
else  
follow the instructions at the end of T-210a
T-210a E210 Incrementations

Four incrementation variables are specified here for use with the T-210 record looping functions.

<table>
<thead>
<tr>
<th>INC1</th>
<th>INC2</th>
<th>INC3</th>
<th>INC4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

INC1 incrementation variable for use with the N1 (node 1) variable on T-210
INC2 incrementation variable for use with the N2 (node 2) variable on T-210
INC3 incrementation variable for use with the N3 (node 3) variable on T-210
INC4 incrementation variable for use with USERELT

Any of these incrementation variables can be negative, zero or positive.

if ( the user wants to add another set of E210 elements ) then
    go to T-210
elseif ( the user wants to add a set of E250 elements ) then
    go to T-250
else
    return to T-100
endif
T-250 Additional E250 Elements

T-250 is the first record of the two-record T-250/T-250a Ecom protocol extension of the T-2 record that also defines an E250 planar boundary condition element. With this record (and T-250a), the user can specify one or more additional planar boundary condition elements to be included in the current element unit. The looping capability provided here, via $\text{nx}$ and the four incrementation parameters on T-250a, can be used effectively in many situations.

A moving plane boundary—which is like a symmetry boundary, except that the symmetry plane itself is allowed to move as a rigid body—is defined in a STAGS element unit with a set of T-250 planar boundary condition elements that are modeled with “beams” that are strung along a user-defined space curve forming the boundary. It is the user’s responsibility to make sure that curve initially lies in a plane. If the constraint is violated initially, STAGS will complain. The reference node $\text{n3}$ must also lie in the boundary plane.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>KELT</th>
<th>ICROSS</th>
<th>XSI</th>
<th>ECY</th>
<th>ECZ</th>
<th>ILIN</th>
<th>IPLAS</th>
<th>NX</th>
<th>USERELT</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>active beam node # 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>active beam node # 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>node # 3 — reference node used to generate the element coordinate system; $\text{n3}$ must lie in the boundary plane</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KELT</td>
<td>must = 250</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICROSS</td>
<td>must = -1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XSI</td>
<td>ignored</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECY</td>
<td>scale factor used for the numerical conditioning of Lagrange constraints introduced by the multipoint constraint; should be of the same order of magnitude as entries in the stiffness matrix; a good guess is the modulus of elasticity of the material</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECZ</td>
<td>ignored</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ILIN</td>
<td>ignored</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPLAS</td>
<td>ignored</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NX</td>
<td>number of planar boundary condition elements to be generated via this T-250 record; set to unity by STAGS if nonpositive or omitted</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USERELT</td>
<td>user-specified element number, used only if $\text{IUWLE} = 1$ on H-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

if ( $\text{nx} > 1$ ) then go to T-250a

else follow the instructions at the end of T-250a
**T-250a E250 Incrementations**

Four incrementation variables are specified here for use with the T-250 record looping functions.

<table>
<thead>
<tr>
<th>INC1</th>
<th>INC2</th>
<th>INC3</th>
<th>INC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>incrementation variable for use with the ( N_1 ) (node 1) variable on T-250</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation variable for use with the ( N_2 ) (node 2) variable on T-250</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation variable for use with the ( N_3 ) (node 3) variable on T-250</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incrementation variable for use with \texttt{USERELT}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero or positive.

```plaintext
if ( the user wants to add another set of \texttt{E250} elements ) then
go to T-250
elseif ( the user wants to add a set of \texttt{E210} elements ) then
go to T-210
else
    return to T-100
endif
```
9.3 Definition of Triangular Elements via the Ecom Protocol

There are two triangular thin shell elements available in the current version of STAGS.

The 3-node \textbf{E320} element is a traditional triangular thin shell element. One or more \textbf{E320} elements (or additional \textbf{E320} elements) can be defined with the T-320 record set. For more information about the \textbf{E320} element, see “E320 Triangular shell element” on page 14-20.

The 3-node \textbf{E330} element is STAGS’ implementation of the MIN3 triangle element in the COMET program. One or more \textbf{E330} elements (or additional \textbf{E330} elements) can be defined with the T-330 record set. For more information about the \textbf{E330} element, see “E330 Triangular shell element” on page 14-21.
T-320 Additional E320 Elements

T-320 is the first record of the multi-record Ecom protocol extension of the T-3/T-3a record set that also defines an E320 triangular shell element. With this record (and T-320a, T-320b and/or T-320c companion records, as required), the user can specify one or more additional E320 triangular shell elements to be included in the current element unit. The looping capabilities provided here, via NX, NY, and the incrementation parameters on T-320a and T-320b, can be used effectively in many situations.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>KELT</th>
<th>IWALL</th>
<th>ZETA</th>
<th>ECZ</th>
<th>ILIN</th>
<th>IPLAS</th>
<th>IANG</th>
<th>USERELT</th>
<th>NX</th>
<th>NY</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>node 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>node 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>node 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KELT</td>
<td>must = 320</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IWALL</td>
<td>wall fabrication identifier:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>shell wall properties are given in user-written subroutine WALL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;0</td>
<td>wall fabrication number, in the Wall Fabrication Table (K-1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When IWALL = 0, data below indicated by ✤ are defined in WALL and are initialized to zero before WALL is called, superseding any values input here

✤ ZETA angle $\zeta$ between the wall-reference coordinate $x_w$ and the fabrication coordinate $x; \ zeta$ is a right-handed rotation about $\tilde{z}$. see Figure 8.2 on page 8-19.

✤ ECZ eccentricity in $z'$ direction. ECZ is the $z'$ coordinate of the shell wall middle surface; see Figure 6.2 on page 6-28. In element units, $(x', y', z')$ are used in place of $(X', Y', Z')$, which do not exist in an element unit. See “Effects of Eccentricity” on page 16-6.

✤ ILIN governs geometric nonlinearity

| 0  | nonlinear strain-displacement relations |
| 1  | linear strain-displacement relations |
IPLAS  
gevens material nonlinearity
0  – elastic behavior only
1  – plasticity included, with the material law satisfied at each element integration point
2  – plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)

IANG  
wall-reference option; see discussion on page 8-18
0  – use default strategy of projecting $x_g, y_g$ to establish $x_w$
1  – input $r_w$, which is projected to establish $x_w$; see (T-320c)

USERELT  
user-specified element number, used only when IUWLE=1 on H-1

if (NX > 1) then go to T-320a
elseif (NY > 1) then go to T-320b
elseif (IANG = 1) then go to T-320c
else follow the instructions at the end of T-320c
**T-320a E320 X-Direction Incrementations**

Four x-direction incrementation variables are specified here for use with the T-320 record looping functions.

<table>
<thead>
<tr>
<th>IX1</th>
<th>IX2</th>
<th>IX3</th>
<th>IX4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ix1</td>
<td>x-direction incrementation variable for node N1 on T-320</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ix2</td>
<td>x-direction incrementation variable for node N2 on T-320</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ix3</td>
<td>x-direction incrementation variable for node N3 on T-320</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ix4</td>
<td>x-direction incrementation variable for use with USERELT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero or positive.

```plaintext
if ( NY > 1 ) then go to T-320b
elseif ( IANG = 1 ) then go to T-320c
else follow the instructions at the end of T-320c
```
**T-320b E320 Y-Direction Incrementations**

Four y-direction incrementation variables are specified here for use with the T-320 record looping functions.

<table>
<thead>
<tr>
<th>IY1</th>
<th>IY2</th>
<th>IY3</th>
<th>IY4</th>
</tr>
</thead>
<tbody>
<tr>
<td>y-direction incrementation variable for node N1 on T-320</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y-direction incrementation variable for node N2 on T-320</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y-direction incrementation variable for node N3 on T-320</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y-direction incrementation variable for use with USERELT</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero or positive.

```plaintext
if ( IANG = 1 ) then go to T-320c
else follow the instructions at the end of T-320c
```
T-320c E320 Wall Reference Vector

RX, RY, RZ

RX, RY, RZ wall reference vector, $r_w$, which is projected onto the element surface to determine the direction of the wall-reference coordinate $x_w$; $r_w$ is expressed in $(x^g, y^g, z^g)$ global coordinates.

if ( the user wants to add another set of E320 elements ) then
  go to T-320
elseif ( the user wants to add a set of E330 elements ) then
  go to T-330
else
  return to T-100
endif
**T-330 Additional E330 Elements**

T-330 is the first record of the multi-record Ecom protocol extension of the T-3/T-3a record set that also defines an E330 triangular shell element. With this record (and T-330a, T-330b and/or T-330c companion records, as required), the user can specify one or more additional E330 shell elements to be included in the current element unit. The *looping* capabilities provided here, via **NX**, **NY**, and the incrementation parameters on T-330a and T-330b, can be used effectively in many situations.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>KELT</th>
<th>IWALL</th>
<th>ZETA</th>
<th>ECZ</th>
<th>ILIN</th>
<th>IANG</th>
<th>USERELT</th>
<th>NX</th>
<th>NY</th>
</tr>
</thead>
<tbody>
<tr>
<td>node 1</td>
<td>node 2</td>
<td>node 3</td>
<td>( KELT ) must = 330</td>
<td>wall fabrication identifier:</td>
<td>angle ( \zeta ) between the wall-reference coordinate ( x_w ) and the fabrication coordinate ( x ); ( \zeta ) is a right-handed rotation about ( z ). see Figure 8.2 on page 8-19.</td>
<td>eccentricity in ( z' ) direction. ( ECZ ) is the ( z' ) coordinate of the shell wall middle surface; see Figure 6.2 on page 6-28. In element units, ((x', y', z')) are used in place of ((X', Y', Z')), which do not exist in an element unit.</td>
<td>governs geometric nonlinearity</td>
<td>0 – nonlinear strain-displacement relations</td>
<td>1 – linear strain-displacement relations</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When **IWALL** = 0, data below indicated by ✤ are defined in subroutine WALL and are set to zero before WALL is called, superseding any values input here.

**ZETA**: angle \( \zeta \) between the wall-reference coordinate \( x_w \) and the fabrication coordinate \( x \); \( \zeta \) is a right-handed rotation about \( z \). see Figure 8.2 on page 8-19.

**ECZ**: eccentricity in \( z' \) direction. \( ECZ \) is the \( z' \) coordinate of the shell wall middle surface; see Figure 6.2 on page 6-28. In element units, \((x', y', z')\) are used in place of \((X', Y', Z')\), which do not exist in an element unit.

**ILIN**: governs geometric nonlinearity

0 – nonlinear strain-displacement relations

1 – linear strain-displacement relations
IPLAS

- governs material nonlinearity
  0 - elastic behavior only
  1 - plasticity included, with the material law satisfied at each element integration point
  2 - plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)

IANG

- wall-reference option; see discussion on page 8-18
  0 - use default strategy of projecting \( x_g, y_g \) to establish \( x_w \)
  1 - input \( r_w \), which is projected to establish \( x_w \); see (T-330c)

USERELT

- user-specified element number, used only when \( \text{IUWLE} = 1 \) on H-1

\[
\begin{align*}
\text{if } & (\text{NX} > 1) \quad \text{then go to T-330a} \\
\text{elseif } & (\text{NY} > 1) \quad \text{then go to T-330b} \\
\text{elseif } & (\text{IANG} = 1) \quad \text{then go to T-330c} \\
\text{else} & \quad \text{follow the instructions at the end of T-330c}
\end{align*}
\]
**T-330a E330 X-Direction Incrementations**

Four x-direction incrementation variables are specified here for use with the T-330 record looping functions.

<table>
<thead>
<tr>
<th>IX1</th>
<th>IX2</th>
<th>IX3</th>
<th>IX4</th>
</tr>
</thead>
<tbody>
<tr>
<td>IX1</td>
<td>x-direction incrementation variable for node N1 on T-330</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX2</td>
<td>x-direction incrementation variable for node N2 on T-330</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX3</td>
<td>x-direction incrementation variable for node N3 on T-330</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX4</td>
<td>x-direction incrementation variable for use with USERELT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero or positive.

```plaintext
if ( NY > 1 ) then go to T-330b
elseif ( IANG = 1 ) then go to T-330c
else follow the instructions at the end of T-330c
```
## T-330b E330 Y-Direction Incrementations

Four y-direction incrementation variables are specified here for use with the T-330 record looping functions.

<table>
<thead>
<tr>
<th>IY1</th>
<th>IY2</th>
<th>IY3</th>
<th>IY4</th>
</tr>
</thead>
<tbody>
<tr>
<td>y-direction incrementation variable for node N1 on T-330</td>
<td>y-direction incrementation variable for node N2 on T-330</td>
<td>y-direction incrementation variable for node N3 on T-330</td>
<td>y-direction incrementation variable for use with USERELT</td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero or positive.

```plaintext
if ( IANG = 1 ) then go to T-330c
else follow the instructions at the end of T-330c
```
### T-330c E330 Wall Reference Vector

<table>
<thead>
<tr>
<th>RX</th>
<th>RY</th>
<th>RZ</th>
</tr>
</thead>
</table>

**RX, RY, RZ**  
Wall reference vector, $r_w$, which is projected onto the element surface to determine the direction of the wall-reference coordinate $x_w$; $r_w$ is expressed in $(x_g, y_g, z_g)$ global coordinates.

- if (the user wants to add another set of E330 elements) then
  - go to T-330
- elseif (the user wants to add a set of E320 elements) then
  - go to T-320
- else
  - return to T-100
endif
There are five “standard” quadrilateral thin shell elements available in the current version of STAGS. Three of these (E410, E411 and E480) are traditional 4- and 9-node quadrilateral elements. The other two (E510 and E710) are 5- and 7-node constructions (of two and four E410s, respectively) that are most appropriately used in modeling transition zones where mesh refinements are required.

The “standard” 4-node E410 element is a traditional thin shell quadrilateral element. One or more E410 elements (or additional E410 elements) can be defined with the T-410 record set—as described on the pages immediately following these introductory remarks. For more information about the E410 element, please see “E410 4–Node quadrilateral shell element” on page 14-22.

The “standard” E411 element is a thin shell quad element that is defined by its four corner node points but which has additional (program-generated) mid-side nodes with supplementary (program-generated) degrees of freedom. One or more E411 elements (or additional E411 elements) can be defined with the T-411 record set—as described later in this section. For more information about the E411 element, please see “E411 4–Node quadrilateral shell element” on page 14-23.

The “standard” 9-node E480 element is a traditional higher-order thin shell quadrilateral element. One or more E480 elements (or additional E480 elements) can be defined with the T-480 record set—as described later in this section. For more information about the E480 element, please see “E480 9–Node quadrilateral shell element” on page 14-24.

The “standard” 5-node E510 “quadrilateral” element has a (user-defined) mid-side node on one of its four edges. It is typically used to connect two 4-node quadrilateral shell elements in a finely-meshed region to a single 4-node quad shell element in another (more-coarsely-meshed) region. One or more E510 elements (or additional E510 elements) can be defined with the T-510 record set—as described later in this section.

The “standard” 7-node E510 “quadrilateral” element has one (user-defined) mid-side node on each of two adjacent edges and an extra (user-defined) internal node. It is typically used to connect 4-node quads in a finely-meshed region to 4-node quads in two or three other (more-coarsely-meshed) regions—at a corner where all of these regions come together. One or more E710 elements (or additional E710 elements) can be defined with the T-710 record set—as described later in this section.
There is one user-developed quadrilateral element in the current version of STAGS.

The 4-node **E940** MIN4 quad element was originally implemented as a User-Defined STAGS element using the STAGS UEL methodologies and standards that are described in Section 9.8 of this chapter and in Chapter 13 of this document. In 2008, this **E940** element was integrated into the current version of STAGS as a “built-in” UEL that can be used by anyone who needs or wants to use that element. One or more **E940** elements can be defined with the T-900 record and the T-940 record set that are also described in Section 9.8 of this chapter. For more information about the **E940** element, please see Chapter 14 in this document, the STAGS Elements Manual, and the STAGS Test Cases Manual.
T-410 Additional E410 Elements

T-410 is the first record of the multi-record Ecom protocol extension of the T-4/T-4b record set that also defines an E410 4-node quadrilateral shell element. With this record (and T-410a, T-410b and/or T-410c companion records, as required), the user can specify one or more additional E410 shell elements to be included in the current element unit. The looping capabilities provided here, via NX, NY, and the incrementation parameters on T-410a and T-410b, can be used effectively in many situations.

A quadrilateral shell element is defined by specifying the four corner nodes in counterclockwise order as viewed from above. In this context, “viewed from above” means looking down onto the top surface of the element. As shown in Figure 6.2 on page 6-28, the top surface corresponds to \( Z' = Z_{\text{max}} \) (\( z' = z'_{\text{max}} \) in an element unit). The four nodes that are input on T-410 determine the \((x', y', z')\) element coordinate system.

The \( z' \) axis defines the direction in which pressure acts—a positive pressure value acts in the positive \( z' \) direction, and a negative pressure value acts in the negative \( z' \) direction.

In shell units, the wall-fabrication orientation is determined by ZETA & ECZ (M-5). Figure 6.2 shows how ECZ defines the eccentricity and how ZETA rotates the \((x, y)\) fabrication coordinates to establish their directions relative to the \((x_w, y_w)\) wall-reference coordinates. Implicit in Figure 6.2 is the shell-unit convention that the wall-reference coordinates are defined to be coincident with the \((X', Y')\) shell coordinates. In element units, shell coordinate systems do not exist. Instead, the user is given two options for establishing wall-reference coordinates.

In the default option, STAGS determines whether the \( x_g \) or \( y_g \) axis lies closer to the element plane. If \( x_g \) lies closer, it is projected onto the element surface to establish the \( x_w \) axis. If \( y_g \) lies closer, it is projected to establish the \( y_w \) axis. This means that when \( x_g \) and \( y_g \) both lie in the plane of the element, the result will be the same regardless of which is chosen to be projected.

The other option is to input a wall reference vector, \( r_w \), that determines the direction of \( x_w \). Although the user will in most cases try to make sure that this vector is tangent to the surface he has in mind, the code makes no such assumption. STAGS will project this vector onto the element surface to establish the \( x_w \) axis.

After the \( x_w \) axis is established, using either of the two options, the wall orientation is then determined just as it is for shell units. Compare Figure 8.2 with Figure 6.2. ZETA has the same meaning in both instances. The only difference is that for shell units, the \((x_w, y_w)\) wall-reference
coordinates are uniquely defined by the \((X', Y')\) shell coordinates, and for element units, one of
two user-selected options (\textit{IANG}) is used to establish the wall-reference system.

This process often allows the user to specify all the wall angles with very simple input. For
example, some standard geometries such as cylinders or cones have an axis of revolution (the
generator) whose projection onto the element surface lies along the same direction, independent
of the element location. If user input specifies the generator as \(r_w\), or if the default is used (for
cone angles less than \(45^\circ\)), a unique angle \textit{ZETA} gives the proper offset for all elements in the
geometry. Figure 8.3 on page 8-20 shows an example where the generator of a cone is specified
as \(r_w\). It is usually straightforward to specify the global coordinates of a vector parallel to a
shell-of-revolution generator. The user should be careful for more complex geometries. It should
also be noted that the user need not know the \((x', y', z')\) shell element coordinate system.

For all quadrilaterals, the plane of the element is determined by the procedure outlined in Section
14.2 “Algorithm for Determining the Element Frame”. The result of this procedure is an element
reference plane that is the “best fit” to the four corner nodes. \((x_w, y_w)\) wall-reference coordinates,
and hence \((\bar{x}, \bar{y})\) fabrication coordinates and \((\phi_1, \phi_2)\) material coordinates, all lie in this “average”
plane.

Note that the fabrication coordinate system is oriented by rotating the wall-
reference system through the angle \textit{ZETA}. The material coordinate system for
each layer in a laminate is then determined by rotating the fabrication
system through a unique angle \textit{ZETL} (K-2) (or \textit{ANGSHL} (I-21d) for GCP input)
for the corresponding layer.

\textbf{N1 N2 N3 N4 KELT IWALL ZETA ECZ ILIN IPLAS INTEG IPENL IANG USERELT NX NY}

\begin{itemize}
  \item \textit{KELT} must = 410
  \item \textit{IWALL} wall fabrication identifier:
    \begin{itemize}
      \item 0 – shell wall properties are given in user-written subroutine \text{WALL}
      \item >0 – shell wall fabrication number in the \text{Wall Fabrication Table} (K-1)
      \item <0 – shell wall fabrication identifier in the \text{GCP Fabrications Table} (I-21a)
    \end{itemize}
\end{itemize}
When \( \text{IWALL} = 0 \), data below indicated by ✤ are defined in WALL; they are automatically initialized to zero before WALL is called, thereby superseding any nonzero values input here.

✦ **ZETA**

angle \( \zeta \) between the wall-reference coordinate \( x_w \) and the fabrication coordinate \( x \); \( \zeta \) is a right-handed rotation about \( z \); see Figure 8.2 on page 8-19

✦ **ECZ**
eccentricity in \( z' \) direction. \( \text{ECZ} \) is the \( z' \) coordinate of the shell wall middle surface; refer to Figure 6.2 on page 6-28; in element units, \( (x', y', z') \) are used in place of \( (X', Y', Z') \), which do not exist in an element unit; see “Effects of Eccentricity” on page 16-6

✦ **ILIN**
governs geometric nonlinearity

0 – nonlinear strain-displacement relations
1 – linear strain-displacement relations

Note that with \( \text{ILIN} = 1 \), bifurcation buckling is suppressed in the shell unit

✦ **IPLAS**
governs material nonlinearity

0 – elastic behavior only
1 – plasticity included, with the material law satisfied at each element integration point
2 – plasticity included, with the material law satisfied at the element centroid (centroplasticity)

✦ **INTEG**
integration type (see N-1)

0 – standard integration, or \( 2 \times 2 \) Gauss points
1 – modified 5-point integration, previously referred to as full integration

✦ **IPENL**
penalty option (see N-1)

0 – no penalty function on fourth-order terms
1 – penalty function included in elements

✦ **IANG**
wall-reference option; see discussion on page 8-18

0 – use default strategy of projecting \( x_g, y_g \) to establish \( x_w \)
1 – input \( r_w \), which is projected to establish \( x_w \); see (T-410c)

✦ **USERELT**
user-specified element number, used only when \( \text{IUWLE} = 1 \) on H-1

\[
\text{if} \quad ( \text{NX} > 0 ) \quad \text{then go to T-410a}
\text{elseif} \quad ( \text{NY} > 0 ) \quad \text{then go to T-410b}
\text{elseif} \quad ( \text{IANG} = 1 ) \quad \text{then go to T-410c}
\text{else} \quad \text{follow the instructions at the end of T-410c}
\]
**T-410a E410 X-Direction Incrementations**

Five x-direction incrementation variables are specified here for use with the T-410 record looping functions (if and only if $\mathbf{NX} > 0$ on T-410).

<table>
<thead>
<tr>
<th>IX1</th>
<th>IX2</th>
<th>IX3</th>
<th>IX4</th>
<th>IX5</th>
</tr>
</thead>
<tbody>
<tr>
<td>IX1</td>
<td>x-direction incrementation variable for node $\mathbf{N1}$ on T-410</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX2</td>
<td>x-direction incrementation variable for node $\mathbf{N2}$ on T-410</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX3</td>
<td>x-direction incrementation variable for node $\mathbf{N3}$ on T-410</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX4</td>
<td>x-direction incrementation variable for node $\mathbf{N4}$ on T-410</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX5</td>
<td>x-direction incrementation variable for use with USERELT</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero or positive.

```plaintext
if ( NY > 1 ) then go to T-410b
elseif ( IANG = 1 ) then go to T-410c
else follow the instructions at the end of T-410c
```
T-410b E410 Y-Direction Incrementations

Five y-direction incrementation variables are specified here for use with the T-410 record looping functions (if and only if NY > 0 on T-410).

<table>
<thead>
<tr>
<th>IY1</th>
<th>IY2</th>
<th>IY3</th>
<th>IY4</th>
<th>IY5</th>
</tr>
</thead>
<tbody>
<tr>
<td>IY1</td>
<td>y-direction incrementation variable for node N1 on T-410</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IY2</td>
<td>y-direction incrementation variable for node N2 on T-410</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IY3</td>
<td>y-direction incrementation variable for node N3 on T-410</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IY4</td>
<td>y-direction incrementation variable for node N4 on T-410</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IY5</td>
<td>y-direction incrementation variable for use with USERELT</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero or positive.

if ( IANG = 1 ) then go to T-410c
else follow the instructions at the end of T-410c
**T-410c E410 Wall Reference Vector**

**RX, RY, RZ**

wall reference vector, $\mathbf{r}_w$, which is projected onto the element surface to determine the direction of the wall-reference coordinate $x_w$;
$\mathbf{r}_w$ is expressed in $(x_g, y_g, z_g)$ global coordinates.

---

if (the user wants to add another set of E410 elements) then
  go to T-410
elseif (the user wants to add a set of E411 elements) then
  go to T-411
elseif (the user wants to add a set of E480 elements) then
  go to T-480
elseif (the user wants to add a set of E510 elements) then
  go to T-510
elseif (the user wants to add a set of E710 elements) then
  go to T-710
else
  return to T-100
endif
**T-411 Additional E411 Elements**

T-411 is the first record of the multi-record Ecom protocol extension of the T-4/T-4b/T-4c record set that also defines an E411 quadrilateral shell element. With this record (and T-411a, T-411b and/or T-411c companion records, as required), the user can specify one or more additional E411 shell elements to be included in the current element unit. The looping capabilities provided here, via NX, NY, and the incremenetation parameters on T-411a and T-411b, can be used effectively in many situations.

For definition purposes, the E411 element is considered to be a 4-node quadrilateral, similar to the E410 element. STAGS adds midside deviational nodes to the E411 element automatically, without user input.

The methods that STAGS uses for determining the element coordinate system for this and other quadrilateral elements are described earlier in this document, in connection with the T-4 and T-410 element-definition records, and will not be repeated here.

---

**N1 N2 N3 N4 KELT IWALL ZETA ECZ ILIN IPLAS INTEG IPENL IANG USERELT NX NY**

N1 node 1  
N2 node 2  
N3 node 3  
N4 node 4  
KELT must = 411  
IWALL wall fabrication identifier:  
0 – shell wall properties are given in user-written subroutine WALL  
>0 – shell wall fabrication number in the Wall Fabrication Table (K-1)  
When IWALL = 0, data below indicated by ✤ are defined in WALL; they are automatically initialized to zero before WALL is called, thereby superseding any nonzero values input here.  

 faucet  
ZETA  
angle zeta between the wall-reference coordinate xw and the fabrication coordinate xi; zeta is a right-handed rotation about z; see Figure 8.2 on page 8-19  

drop  
ECZ  
eccentricity in z' direction; ECZ is the z' coordinate of the shell wall middle surface; refer to Figure 6.2 on page 6-28; in element units, (x', y', z') are used in place of (x', y', Z') , which do not exist in an element unit.  

drop  
ILIN governs geometric nonlinearity  
0 – nonlinear strain-displacement relations  
1 – linear strain-displacement relations
Note that with $\text{ILIN} = 1$, bifurcation buckling is suppressed in the shell unit.

**IPLAS**
- governs material nonlinearity
  - 0 – elastic behavior only
  - 1 – plasticity included, with the material law satisfied at each element integration point
  - 2 – plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)

**INTEG**
- integration type (see N-1)
  - 0 – standard integration, or $2 \times 2$ Gauss points
  - 1 – modified 5-point integration, previously referred to as full integration

**IPENL**
- penalty option (see N-1)
  - 0 – no penalty function on fourth-order terms
  - 1 – penalty function included in elements

**IANG**
- wall-reference option; see discussion on page 8-18
  - 0 – use default strategy of projecting $x'_{g}, y'_{g}$ to establish $x_{w}$
  - 1 – input $r_{w}$, which is projected to establish $x_{w}$; see (T-411c)

**USERELT**
- user-specified element number, used only when $\text{IUWLE} = 1$ on H-1

```plaintext
if (NX > 0) then go to T-411a
elseif (NY > 0) then go to T-411b
elseif (IANG = 1) then go to T-411c
else follow the instructions at the end of T-411c
```
T-411a E411 X-Direction Incrementations

Five x-direction incrementation variables are specified here for use with the T-411 record looping functions (if and only if $\text{NX} > 0$ on T-411).

<table>
<thead>
<tr>
<th>IX1</th>
<th>IX2</th>
<th>IX3</th>
<th>IX4</th>
<th>IX5</th>
</tr>
</thead>
<tbody>
<tr>
<td>IX1</td>
<td>x-direction incrementation variable for node $\text{N1}$ on T-411</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX2</td>
<td>x-direction incrementation variable for node $\text{N2}$ on T-411</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX3</td>
<td>x-direction incrementation variable for node $\text{N3}$ on T-411</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX4</td>
<td>x-direction incrementation variable for node $\text{N4}$ on T-411</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX5</td>
<td>x-direction incrementation variable for use with $\text{USERELT}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero or positive.

```
if ( $\text{NY} > 1$ ) then go to T-411b
elseif ( $\text{IANG} = 1$ ) then go to T-411c
else follow the instructions at the end of T-411c
```
**T-411b E411 Y-Direction Incrementations**

Five y-direction incrementation variables are specified here for use with the T-411 record looping functions (if and only if NY > 0 on T-411).

<table>
<thead>
<tr>
<th>IY1</th>
<th>IY2</th>
<th>IY3</th>
<th>IY4</th>
<th>IY5</th>
</tr>
</thead>
<tbody>
<tr>
<td>IY1</td>
<td>y-direction incrementation variable for node N1 on T-411</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IY2</td>
<td>y-direction incrementation variable for node N2 on T-411</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IY3</td>
<td>y-direction incrementation variable for node N3 on T-411</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IY4</td>
<td>y-direction incrementation variable for node N4 on T-411</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IY5</td>
<td>y-direction incrementation variable for use with USERELT</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero or positive.

if ( IANG = 1 ) then go to T-411c
else follow the instructions at the end of T-411c
**T-411c E411 Wall Reference Vector**

**RX  RY  RZ**

- **RX, RY, RZ**: Wall reference vector, $r_w$, which is projected onto the element surface to determine the direction of the wall-reference coordinate $x_w$. $r_w$ is expressed in $(x_w^g, y_w^g, z_w^g)$ global coordinates.

```plaintext
if ( the user wants to add another set of E411 elements ) then
go to T-411
elseif ( the user wants to add a set of E410 elements ) then
go to T-410
elseif ( the user wants to add a set of E480 elements ) then
go to T-480
elseif ( the user wants to add a set of E510 elements ) then
go to T-510
elseif ( the user wants to add a set of E710 elements ) then
go to T-710
else
  return to T-100
endif
```
**T-480 Additional E480 Elements**

T-480 is the first record of the multi-record Ecom protocol extension of the T-4/T-4b/T-4c record set that also defines an **E480** 9-node quadrilateral shell element. With this record and T-480a (and T-480b, T-480c and/or T-480d companion records, as required), the user can specify one or more additional **E480** 9-node quadrilateral elements to be included in the current element unit. The *looping* capabilities provided here, via **NX**, **NY** and the incrementation parameters on T-480b and T-480c, can be used effectively in many situations.

The methods that **STAGS** uses for determining the element coordinate system for this and other quadrilateral elements are described earlier in this document, in connection with the T-4 and T-410 element-definition records, and will not be repeated here.

---

**N1 N2 N3 N4 KELT IWALL ZETA ECZ ILIN IPLAS INTEG IPENL IANG USERELT NX NY**

- **N1**: node 1
- **N2**: node 2
- **N3**: node 3
- **N4**: node 4
- **KELT**: must = 480
- **IWALL**: wall fabrication identifier:
  - 0 – shell wall properties are given in user-written subroutine WALL
  - >0 – shell wall fabrication number in the Wall Fabrication Table (K-1)
  - <0 – shell wall fabrication identifier in the GCP Fabrications Table (I-21a)

When **IWALL** = 0, data below indicated by ✤ are defined in WALL; they are automatically initialized to zero before WALL is called, thereby superseding any nonzero values input here.

- ✤ **ZETA**: angle \( \zeta \) between the wall-reference coordinate \( x_w \) and the fabrication coordinate \( x \); \( \zeta \) is a right-handed rotation about \( z \).
- ✤ **ECZ**: eccentricity in \( z' \) direction. **ECZ** is the \( z' \) coordinate of the shell wall middle surface; in element units, \( (x', y', z') \) are used in place of \( (X', Y', Z') \), which do not exist in an element unit.
- ✤ **ILIN**: governs geometric nonlinearity
  - 0 – nonlinear strain-displacement relations
  - 1 – linear strain-displacement relations

Note that with **ILIN** = 1, bifurcation buckling is suppressed in the shell unit.
**IPLAS**

- governs material nonlinearity
- 0 – elastic behavior only
- 1 – plasticity included, with the material law satisfied at each element integration point
- 2 – plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)

**INTEG**

- integration type: not used for this element; set \( \text{INTEG} = 0 \)

**IPENL**

- penalty option: not used for this element; set \( \text{IPENL} = 0 \)

**IANG**

- wall-reference option; see discussion on page 8-18
- 0 – use default strategy of projecting \( x_g,y_g \) to establish \( x_w \)
- 1 – input \( r_w \), which is projected to establish \( x_w \); see (T-480d)

**USERELT**

- user-specified element number, used only when \( \text{IUWLE} = 1 \) on H-1

*go to* T-480a
**T-480a E480 Extra Nodes Specification**

This record is required to specify the fifth, sixth, seventh, eighth and ninth nodes of the (first) E480 element that is defined via the current T-480 record.

<table>
<thead>
<tr>
<th>N5</th>
<th>N6</th>
<th>N7</th>
<th>N8</th>
<th>N9</th>
</tr>
</thead>
<tbody>
<tr>
<td>node 5</td>
<td>node 6</td>
<td>node 7</td>
<td>node 8</td>
<td>node 9</td>
</tr>
</tbody>
</table>

if \( (N_X > 0) \) then go to T-480b
elseif \( (N_Y > 0) \) then go to T-480c
elseif \( (IANG = 1) \) then go to T-480d
else follow the instructions at the end of T-480d
### T-480b E480 X-Direction Incrementations

Ten x-direction incrementation variables are specified here for use with the T-480 record looping functions (if and only if $\mathbf{nx} > 0$ on T-480).

<table>
<thead>
<tr>
<th>IX1</th>
<th>IX2</th>
<th>IX3</th>
<th>IX4</th>
<th>IX5</th>
<th>IX6</th>
<th>IX7</th>
<th>IX8</th>
<th>IX9</th>
<th>IXU</th>
</tr>
</thead>
<tbody>
<tr>
<td>ix1</td>
<td>ix2</td>
<td>ix3</td>
<td>ix4</td>
<td>ix5</td>
<td>ix6</td>
<td>ix7</td>
<td>ix8</td>
<td>ix9</td>
<td>ixu</td>
</tr>
</tbody>
</table>

- **IX1**: x-direction incrementation variable for node $\mathbf{N1}$ on T-480
- **IX2**: x-direction incrementation variable for node $\mathbf{N2}$ on T-480
- **IX3**: x-direction incrementation variable for node $\mathbf{N3}$ on T-480
- **IX4**: x-direction incrementation variable for node $\mathbf{N4}$ on T-480
- **IX5**: x-direction incrementation variable for node $\mathbf{N5}$ on T-480a
- **IX6**: x-direction incrementation variable for node $\mathbf{N6}$ on T-480a
- **IX7**: x-direction incrementation variable for node $\mathbf{N7}$ on T-480a
- **IX8**: x-direction incrementation variable for node $\mathbf{N8}$ on T-480a
- **IX9**: x-direction incrementation variable for node $\mathbf{N9}$ on T-480a
- **IXU**: x-direction incrementation variable for use with USERELT

Any of these incrementation variables can be negative, zero or positive.

```plaintext
if ( \( \mathbf{ny} > 1 \) ) then go to T-480c
elseif ( \( \mathbf{iang} = 1 \) ) then go to T-480d
else follow the instructions at the end of T-480d
```
**T-480c E480 Y-Direction Incrementations**

Ten y-direction incrementation variables are specified here for use with the T-480 record looping functions (if and only if \( NY > 0 \) on T-480).

<table>
<thead>
<tr>
<th>IY1</th>
<th>IY2</th>
<th>IY3</th>
<th>IY4</th>
<th>IY5</th>
<th>IY6</th>
<th>IY7</th>
<th>IY8</th>
<th>IY9</th>
<th>IYU</th>
</tr>
</thead>
<tbody>
<tr>
<td>IY1</td>
<td>x-direction incrementation variable for node ( N_1 ) on T-480</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IY2</td>
<td>x-direction incrementation variable for node ( N_2 ) on T-480</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IY3</td>
<td>x-direction incrementation variable for node ( N_3 ) on T-480</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IY4</td>
<td>x-direction incrementation variable for node ( N_4 ) on T-480</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IY5</td>
<td>x-direction incrementation variable for node ( N_5 ) on T-480a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IY6</td>
<td>x-direction incrementation variable for node ( N_6 ) on T-480a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IY7</td>
<td>x-direction incrementation variable for node ( N_7 ) on T-480a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IY8</td>
<td>x-direction incrementation variable for node ( N_8 ) on T-480a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IY9</td>
<td>x-direction incrementation variable for node ( N_9 ) on T-480a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IYU</td>
<td>x-direction incrementation variable for use with USERELT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero or positive.

\[
\text{if } ( \ IANG \ = \ 1 \ ) \ \text{then go to T-480d} \\
\text{else } \ follow \ the \ instructions \ at \ the \ end \ of \ T-411d
\]
**T-480d E480 Wall Reference Vector**

RX, RY, RZ

wall reference vector, \( \mathbf{r}_w \), which is projected onto the element surface to determine the direction of the wall-reference coordinate \( x_w \); \( \mathbf{r}_w \) is expressed in \((x_g, y_g, z_g)\) global coordinates.

If (the user wants to add another set of E480 elements) then 
go to T-480
elseif (the user wants to add a set of E410 elements) then 
go to T-410
elseif (the user wants to add a set of E411 elements) then 
go to T-411
elseif (the user wants to add a set of E510 elements) then 
go to T-510
elseif (the user wants to add a set of E710 elements) then 
go to T-710
else
    return to T-100
endif
**T-510 Additional E510 Elements**

T-510 is the first record of the multi-record Ecom protocol extension of the T-4/T-4b/T-4c record set that also defines an **E510** 5-node quadrilateral mesh transition element. With this record and T-510a (and with T-510b and/or T-510c companion records, as required), the user can specify one or more additional **E510** 5-node quadrilateral mesh transition elements to be included in the current element unit. The *looping* capabilities provided here, via **NX** and the six incrementation parameters on T-510b, can be used effectively in many situations.

The methods that **STAGS** uses for determining the element coordinate system for this and other quadrilateral elements are described earlier in this document, in connection with the T-4 and T-410 element-definition records, and will not be repeated here.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
<th>KELT</th>
<th>IWALL</th>
<th>ZETA</th>
<th>ECZ</th>
<th>IPLAS</th>
<th>INTEG</th>
<th>IPENL</th>
<th>IANG</th>
<th>USERELT</th>
<th>NX</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>node 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>node 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>node 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N4</td>
<td>node 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KELT</td>
<td>must = 510</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IWALL</td>
<td>wall fabrication identifier:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>shell wall properties are given in user-written subroutine WALL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;0</td>
<td>shell wall fabrication number in the <strong>Wall Fabrication Table</strong> (K-1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;0</td>
<td>shell wall fabrication identifier in the <strong>GCP Fabrications Table</strong> (I-21a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When **IWALL** = 0, data below indicated by ✤ are defined in WALL; they are automatically initialized to zero before WALL is called, thereby superseding any nonzero values input here.

- **ZETA**
  angle $\zeta$ between the wall-reference coordinate $x_w$ and the fabrication coordinate $x$; $\zeta$ is a right-handed rotation about $z$.

- **ECZ**
  eccentricity in $z'$ direction. **ECZ** is the $z'$ coordinate of the shell wall middle surface; in element units, $(x', y', z')$ are used in place of $(X', Y', Z')$, which do not exist in an element unit.

- **ILIN**
  governs geometric nonlinearity
  - 0 — nonlinear strain-displacement relations
  - 1 — linear strain-displacement relations

Note that with **ILIN** = 1, bifurcation buckling is suppressed in the shell unit.
**IPLAS**

- governs material nonlinearity
  - 0: elastic behavior only
  - 1: plasticity included, with the material law satisfied at each element integration point
  - 2: plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)

**INTEG**

- integration type (see N-1)
  - 0: standard integration, or $2 \times 2$ Gauss points
  - 1: modified 5-point integration, previously referred to as full integration

**IPENL**

- penalty option (see N-1)
  - 0: no penalty function on fourth-order terms
  - 1: penalty function included in elements

**IANG**

- wall-reference option; see discussion on page 8-18
  - 0: use default strategy of projecting $x'_w, y'_w$ to establish $x_w$
  - 1: input $r_w$, which is projected to establish $x_w$; see (T-510c)

**USERELT**

- user-specified element number, used only when IUWLE = 1 on H-1
**T-510a E510 Extra Node Specification**

This record is required to specify the fifth node of the first E510 element that is defined via the current T-510 record (which defines nx E510 elements if nx > 0).

---

**N5**

N5 node 5

- if (nx > 0) then go to T-510b
- elseif (IANG = 1) then go to T-510c
- else follow the instructions at the end of T-510c
**T-510b E510 Incrementations**

Six incrementation variables are specified here for use with the T-510 record looping functions (if and only if $\mathbf{NX} > 0$ on T-510).

<table>
<thead>
<tr>
<th>IX1</th>
<th>IX2</th>
<th>IX3</th>
<th>IX4</th>
<th>IX5</th>
<th>IXU</th>
</tr>
</thead>
<tbody>
<tr>
<td>IX1</td>
<td>incrementation variable for node N1 on T-510</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX2</td>
<td>incrementation variable for node N2 on T-510</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX3</td>
<td>incrementation variable for node N3 on T-510</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX4</td>
<td>incrementation variable for node N4 on T-510</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX5</td>
<td>incrementation variable for node N5 on T-510a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IXU</td>
<td>incrementation variable for use with USERELT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero or positive.

```
if ( IANG = 1 ) then go to T-510c
else follow the instructions at the end of T-510c
```
**T-510c E510 Wall Reference Vector**

**RX  RY  RZ**

**RX, RY, RZ** wall reference vector, \( \mathbf{r}_w \), which is projected onto the element surface to determine the direction of the wall-reference coordinate \( x_w \); \( \mathbf{r}_w \) is expressed in \((x_g, y_g, z_g)\) global coordinates.

```c
if ( the user wants to add another set of E510 elements ) then
go to T-510
elseif ( the user wants to add a set of E410 elements ) then
go to T-410
elseif ( the user wants to add a set of E411 elements ) then
go to T-411
elseif ( the user wants to add a set of E490 elements ) then
go to T-480
elseif ( the user wants to add a set of E710 elements ) then
go to T-710
else
    return to T-100
endif
```
**T-710 Additional E710 Elements**

T-710 is the first record of the multi-record Ecom protocol extension of the T-4/T-4b/T-4c record set that also defines an **E710 7-node quadrilateral mesh transition element**. With this record and T-710a (and with T-710b and/or T-710c companion records, as required), the user can specify one or more additional **E710 7-node quadrilateral mesh transition elements** to be included in the current element unit. The looping capabilities provided here, via **NX** and the eight incrementation parameters on T-710b, can be used effectively in many situations.

The methods that **STAGS** uses for determining the element coordinate system for this and other quadrilateral elements are described earlier in this document, in connection with the T-4 and T-410 element-definition records, and will not be repeated here.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
<th>KELT</th>
<th>IWALL</th>
<th>ZETA</th>
<th>ECZ</th>
<th>ILIN</th>
<th>IPLAS</th>
<th>INTEG</th>
<th>IPENL</th>
<th>IANG</th>
<th>USERELT</th>
<th>NX</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>N2</td>
<td>N3</td>
<td>N4</td>
<td><strong>KELT</strong></td>
<td><strong>IWALL</strong></td>
<td><strong>ZETA</strong></td>
<td><strong>ECZ</strong></td>
<td><strong>ILIN</strong></td>
<td><strong>IPLAS</strong></td>
<td><strong>INTEG</strong></td>
<td><strong>IPENL</strong></td>
<td><strong>IANG</strong></td>
<td><strong>USERELT</strong></td>
<td><strong>NX</strong></td>
</tr>
</tbody>
</table>

- **N1, N2, N3, N4**: Node numbers
- **KELT**: Wall fabrication identifier; must = 710
- **IWALL**: Wall fabrication identifier:
  - 0: Shell wall properties are given in user-written subroutine WALL
  - >0: Shell wall fabrication number in the Wall Fabrication Table (K-1)
  - <0: Shell wall fabrication identifier in the GCP Fabrications Table (L-21a)

When **IWALL = 0**, data below indicated by ✤ are defined in WALL; they are automatically initialized to zero before WALL is called, thereby superseding any nonzero values input here.

- **ZETA**: Angle ζ between the wall-reference coordinate x_w and the fabrication coordinate x; ζ is a right-handed rotation about z.
- **ECZ**: Eccentricity in z' direction. ECZ is the z' coordinate of the shell wall middle surface; in element units, (x',y',z') are used in place of (X',Y',Z'), which do not exist in an element unit.
- **ILIN**: Governs geometric nonlinearity
  - 0: Nonlinear strain-displacement relations
  - 1: Linear strain-displacement relations

Note that with **ILIN = 1**, bifurcation buckling is suppressed in the shell unit.
**IPLAS**

governs material nonlinearity

0 – elastic behavior only

1 – plasticity included, with the material law satisfied at each element integration point

2 – plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)

**INTEG**

integration type (see N-1)

0 – standard integration, or $2 \times 2$ Gauss points

1 – modified 5-point integration, previously referred to as full integration

**IPENL**

penalty option (see N-1)

0 – no penalty function on fourth-order terms

1 – penalty function included in elements

**IANG**

wall-reference option; see discussion on page 8-18

0 – use default strategy of projecting $x,f,y,g$ to establish $x,w$

1 – input $r,w$, which is projected to establish $x,w$; see (T-710c)

**USERELT**

user-specified element number, used only when $I,UWE = 1$ on H-1

*go to T-710a*
**T-710a E710 Extra Nodes Specification**

This record is required to specify the fifth, six and seventh nodes of the first E710 element that is defined via the current T-710 record (which defines nx E710 elements if nx > 0).

<table>
<thead>
<tr>
<th>N5</th>
<th>node 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>N6</td>
<td>node 6</td>
</tr>
<tr>
<td>N7</td>
<td>node 7</td>
</tr>
</tbody>
</table>

if ( nx > 0 ) then go to T-710b  
elseif ( IANG = 1 ) then go to T-710c  
else follow the instructions at the end of T-710c
**T-710b E710 Incrementations**

Eight incrementation variables are specified here for use with the T-710 record looping functions (if and only if \( nx > 0 \) on T-710).

<table>
<thead>
<tr>
<th>IX1</th>
<th>IX2</th>
<th>IX3</th>
<th>IX4</th>
<th>IX5</th>
<th>IX6</th>
<th>IX7</th>
<th>IXU</th>
</tr>
</thead>
<tbody>
<tr>
<td>IX1 incrementation variable for node N1 on T-710</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX2 incrementation variable for node N2 on T-710</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX3 incrementation variable for node N3 on T-710</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX4 incrementation variable for node N4 on T-710</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX5 incrementation variable for node N5 on T-710a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX6 incrementation variable for node N6 on T-710a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX7 incrementation variable for node N7 on T-710a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IXU incrementation variable for use with USERELT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero or positive.

```
if ( IANG = 1 ) then go to T-710c
else follow the instructions at the end of T-710c
```
**T-710c E710 Wall Reference Vector**

**RX, RY, RZ**

RX, RY, RZ is the wall reference vector, $r_w$, which is projected onto the element surface to determine the direction of the wall-reference coordinate $x_w$; $r_w$ is expressed in $(x_g, y_g, z_g)$ global coordinates.

```plaintext
if ( the user wants to add another set of E710 elements ) then
go to T-710
elseif ( the user wants to add a set of E410 elements ) then
go to T-410
elseif ( the user wants to add a set of E411 elements ) then
go to T-411
elseif ( the user wants to add a set of E480 elements ) then
go to T-480
elseif ( the user wants to add a set of E510 elements ) then
go to T-510
else
  return to T-100
endif
```
9.5 Definition of Contact Elements via the Ecom Protocol

The current version of STAGS has three “elements” that enable the user to perform analyses in which three types of contact might occur.

The E810 PAD surface/surface contact element is designed to treat situations in which one surface may come into contact with another, when the specific elements on one surface that may come into contact with specific elements on the other surface are known \textit{a priori}.

The E820 generalized contact “element” is designed to treat situations in which one surface may come into contact with another surface, but where the specific elements that may come into contact with each other are \textit{not} known in advance and/or when they may change during the course of the analysis to be performed. The E820 approach considers contact to be point/surface phenomenon. Here, the user specifies a set of points on one surface and a set of elements on the other, telling STAGS that one or more of the identified points \textit{may} come into contact one or more of the identified elements. STAGS digests this information and uses it to determine whether or not point/surface contact has started (or is continuing) at each step of the analysis, and generates one or more point/surface contact elements as and if required, for the specific points and surfaces that are involved with that contact.

The E822 line/line contact “element” is designed to treat situations in which one line (edge) may come into contact with another line (edge), but where the exact location at which that contact may occur is not known in advance and/or when it may change during the course of the analysis. The E822 approach considers this type of contact to be line/line phenomenon. Here, the user specifies a set of two or more line segments in the model—and uses an E822 line/line contact “element” to identify one line that \textit{may} come into contact with another, for each line/line contact situation that might occur. STAGS digests this information and uses it to determine whether or not line/line contact has started (or is continuing) at each step of the analysis, and generates one or more line/line contact elements as and if required.

The theoretical foundations of these three types of contact “elements” are described in agonizing detail in the \textit{STAGS Elements Manual} document. The input requirements for these elements are described next.
T-810 E810 PAD Contact Element

The E810 8-node PAD contact element is basically a set of four independent nonlinear springs connecting the nodes defining one E410 shell element to the corresponding nodes of a second E410 element. It is intended for use in situations where those two PAD-connected elements—which may be in the same or in different shell or element units—may come into contact with each other. The E810 PAD element is suitable for use only in situations where the contact region is known a priori, where the individual elements coming into contact with each other can be readily identified and paired, and where no sliding along the contact surface occurs. The PAD element can be used to treat (some) lap-joint and Hertzian impact problems, but it is not designed for use in more general situations where the contact regions are not known a priori or when sliding occurs or when friction is present. STAGS’ more general (E820) contact capabilities must be used in those situations.

![Figure 9.1 E810 PAD Element](image)

The nonlinear spring connecting each pair of nodes typically has a very low stiffness when the gap separating the two nodes is positive (where the gap is defined as the distance of the upper-element node from the lower-element reference plane) and to be very stiff when the gap is extremely small or negative (indicating that at least some portions of the two parent elements are in contact with each other): the large forces and stiffnesses that are produced by stiff PAD-element springs helps to enforce the displacement compatibility constraints for the contact problem.

Stiffness properties for PAD-element springs must be specified via one or more stiffness-displacement tables (type I-4a through I-4d records). These stiffness profiles, each of which is
identified by its *profile-definition table number*, are currently assumed to be independent of velocity (any velocity dependencies specified are ignored by the program).

The *user point* nodes that are employed in defining PAD elements are defined *via* type S-1/S-1a/S-2 record sets (or *via* type S-3/S-3a/S4 record sets): these user points are generally slaved to nodes defining the E410 shell elements that are expected to come in contact with each other. An additional OFFSET parameter can be used to account for thickness effects when the user point nodes lie on parent-node reference surfaces within (rather than on the surfaces of) those parent elements.

The T-810 record defines one or more E810 elements to be included in the current element unit. The *looping* capabilities provided here, *via* NX and on T-810 and the incrementation parameters on T-810a, can be used effectively in many situations.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
<th>N5</th>
<th>N6</th>
<th>N7</th>
<th>N8</th>
<th>KELT</th>
<th>ITAB</th>
<th>OFFSET</th>
<th>NX</th>
<th>USERELT</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>first lower-surface PAD-element node point</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>second lower-surface PAD-element node point</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>third lower-surface PAD-element node point</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N4</td>
<td>fourth lower-surface PAD-element node point</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N5</td>
<td>first upper-surface PAD-element node point, connected to N1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N6</td>
<td>second upper-surface PAD-element node, connected to N2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N7</td>
<td>third upper-surface PAD-element node, connected to N3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N8</td>
<td>fourth upper-surface PAD-element node, connected to N4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KELT</td>
<td>element code number 810 defines an 8-node PAD element</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ITAB</td>
<td>spring stiffness-displacement-table identifier (see record I–4a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OFFSET</td>
<td>offset parameter, to account for element thicknesses</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NX</td>
<td>looping parameter, set equal to unity by STAGS if omitted or nonpositive</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USERELT</td>
<td>user-specified element number, used only if IUWLE =1 on H-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

if (NX > 1) go to T-810a
else return to T-100
**T-810a PAD Element Incrementations**

If the \textbf{nx} looping parameter is greater than 1 on T-810, a T-810a record is required to specify nine incrementation variables to be used with the T-810 looping function.

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>I2</td>
<td>I3</td>
<td>I4</td>
<td>I5</td>
<td>I6</td>
<td>I7</td>
<td>I8</td>
<td>I9</td>
<td></td>
</tr>
</tbody>
</table>

- \textbf{I1} incrementation for the \textbf{N1 (PAD node)} variable on the parent T-810 record
- \textbf{I2} incrementation for the \textbf{N2 (PAD node)} variable on the parent T-810 record
- \textbf{I3} incrementation for the \textbf{N3 (PAD node)} variable on the parent T-810 record
- \textbf{I4} incrementation for the \textbf{N4 (PAD node)} variable on the parent T-810 record
- \textbf{I5} incrementation for the \textbf{N5 (PAD node)} variable on the parent T-810 record
- \textbf{I6} incrementation for the \textbf{N6 (PAD node)} variable on the parent T-810 record
- \textbf{I7} incrementation for the \textbf{N7 (PAD node)} variable on the parent T-810 record
- \textbf{I8} incrementation for the \textbf{N8 (PAD node)} variable on the parent T-810 record
- \textbf{I9} incrementation for \textbf{IUWLE}

Any of these incrementation variables can be negative, zero, or positive, as required.

**Example:** the following T-810/T-810a record combination:

\begin{verbatim}
  10 20 30 40 50 60 70 80  810  7 0.0  3
  1 1 1 1 1 1 1 1
\end{verbatim}

generates the same three PAD elements as the following three individual T-810 records:

\begin{verbatim}
  10 20 30 40 50 60 70 80  810  7 0.0
  11 21 31 41 51 61 71 81  810  7 0.0
  12 22 32 42 52 62 72 82  810  7 0.0 1
\end{verbatim}

*return to T-100*
T-820 General Contact Definition

As noted in the preceding description, E810 PAD elements are defined explicitly and should only be used in situations where particular E410 elements that may contact each other are known a priori and can be paired, and where the contact regions (and E410-element pairings) do not change during an analysis. STAGS’ more general point/surface contact capabilities must be used in situations where contact is anticipated but it is not known where the contact will occur and/or how the contact region changes as the analysis progresses.

It is convenient, here, for the analyst to view the contact problem as one in which one structure is (perhaps arbitrarily) designated as the contacting structure and the other is considered to be the contacted one. The current approach in STAGS to the general contact problem avoids many of the complexities and inefficiencies of general surface-on-surface interactions by considering one or more specific points on one structure that may come into contact with one or more shell elements on the other structure—as shown schematically in Figure 9.1:

![Contacting Structure Diagram](image)

Figure 9.2  Contact-Definition Specification in STAGS

The general-contact capabilities in STAGS are invoked when the analyst, anticipating the possibility that contact between two structural components may occur, includes one or more E820 contact-definition specifications in an element unit of the STAGS model. Each contact definition identifies a set of contact points on the contacting structure that may experience contact with a particular contact surface on the contacted structure—where the contact surface is a set of contiguous shell elements forming a convex region on the surface of the latter body.
STAGS uses these contact-definitions to check for contact and to construct actual contact elements coupling contacting points with contacted shell elements on-the-fly, as and if required, as the analysis progresses. In doing this, STAGS attempts to use penalty functions to enforce displacement-compatibility constraints between each contacting point and each element with which it is in contact—utilizing analyst-supplied stiffness-vs.-displacement information to compute the forces and stiffnesses resulting from the (necessarily) small contact-surface penetrations that occur.

An actual contact element is (conceptually) a nonlinear spring connecting the contacting point to the surface of the contacted element. This nonlinear spring typically has a very low stiffness and generates a small force when the contact-surface penetration is small, but it gets progressively stiffer and generates a larger force as the penetration increases. Stiffness properties for these springs must be specified via one or more stiffness-displacement tables (type I-4a through I-4d records). These stiffness-displacement functions each of which is identified by its table number, are currently assumed to be independent of velocity (any velocity dependencies specified are ignored by the program).

The T-820 record defines a single E820 definition to be included in the current element unit. An E820 contact-definition is made via a T-820 record (described next) that is followed first by as many T-820a or T-820b contact-surface-element-specification records, and then by as many T-820c or T-820d contact-point-specification records as may be required. Each of these contact-surface-specifications references a specific stiffness-displacement (penalty) function table.
KELT   NSRF   NPTS

KELT  element code number 820 identifies a contact-definition specification

NSRF  parameter selecting the method that is to be used to identify the contiguous shell elements defining the contact surface, and (optionally) to specify the minimum number of elements required to define that surface (see Note 1, below):

\[
\text{NSRF} \geq 0 \quad \text{— use the row & column method to identify one or more elements in one or more shell units, or}
\]

\[
\text{NSRF} < 0 \quad \text{— use the element-number method to identify one or more elements in one or more shell units and/or one or more elements in the current or previously-defined element units}
\]

NPTS  parameter selecting the method that is to be used to identify the specific points that may come into contact with the contact surface, and (optionally) to specify the minimum number of points that are to be specified (see Note 2, below):

\[
\text{NPTS} \geq 0 \quad \text{— use the row & column method to identify one or more nodes in one or more shell units, or}
\]

\[
\text{NPTS} < 0 \quad \text{— use the point-number method to identify one or more nodes in one or more shell units and/or in the current or previously-defined element units}
\]

Note 1: In the row & column method of defining contact elements, STAGS reads one or more type T-820a records—each of which specifies one or more elements of the contact surface. STAGS keeps a running count of the number of elements specified by each T-820a record—and stops processing T-820a records as soon as NSRF or more elements have been specified. In the element-number method of defining contact elements, STAGS reads one or more type T-820b records—each of which specifies one or more elements of the contact surface. STAGS keeps a running count of the number of elements specified by each T-820b record—and stops processing T-820b records as soon as \(| \text{NSRF} | \) or more elements have been specified.

Note 2: In the row & column method of defining contact points, STAGS reads one or more type T-820c records—each of which specifies one or more contact points. STAGS keeps a running count of the number of points specified by each T-820c record—and stops processing T-820c records as soon as NPTS or more points have been specified. In the element-number method of defining contact points, STAGS reads one or more type T-820d records—each of which specifies one or more contact points. STAGS keeps a running count of the number of points specified by each T-820d record—and stops processing T-820d records as soon as \(| \text{NPTS} | \) or more points have been specified.

\[
\text{if } ( \text{NSRF} \geq 0 ) \text{ then}
\]

\[
\text{go to T-820a}
\]

\[
\text{else}
\]

\[
\text{go to T-820b}
\]
**T-820a E820 Row & Column Contact-Element Specifications**

**NSRF** ≥ 0 (T-820) indicates that the *row & column method* is to be used to identify the contiguous shell elements that comprise the *contact surface*. This method clearly can only be used to identify contact elements that are associated with one or more *shell* units, because there are no row or column associations for elements in element units. With this method, the value of the **NSRF** parameter indicates the **minimum number** of contact elements to be identified with the set of one or more T-820a records included for the current *contact-definition* (see Note 1 for the T-820 record). The number of contact elements identified by any given T-820a depends on the parameters on that record.

<table>
<thead>
<tr>
<th>USRF</th>
<th>TYPE</th>
<th>LI</th>
<th>LJ</th>
<th>ID</th>
<th>NI</th>
<th>NJ</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>USRF</strong></td>
<td>identifies the <em>shell unit</em> within which the element(s) identified by the <strong>LI</strong> and <strong>LJ</strong> row &amp; column parameters are defined; the element grid for this <em>shell unit</em> has <strong>NROWS</strong> rows and <strong>NCOLS</strong> columns of quadrilateral domains—each of which contains one <strong>E410</strong> quadrilateral element or two <strong>E320</strong> triangular elements.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TYPE</strong></td>
<td>specifies the element type for this (or these) element(s): <strong>TYPE</strong> must currently be either <strong>E410</strong> or <strong>E320</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>LI,LJ</strong></td>
<td>row and column numbers identifying one or more elements: <strong>LI</strong> = <strong>LJ</strong> = 0 indicates that all type <strong>TYPE</strong> elements of shell unit <strong>USRF</strong> are to be included; <strong>LI</strong> &gt; 0 with <strong>LJ</strong> = 0 identifies (and includes) all of the elements in row # <strong>LI</strong> of the element grid; <strong>LJ</strong> &gt; 0 with <strong>LI</strong> = 0 identifies all of the elements in column <strong>LJ</strong>; and <strong>LI</strong> &gt; 0 with <strong>LJ</strong> &gt; 0 identifies one or more <strong>E410</strong> element or the one or more pairs of <strong>E320</strong> elements, starting at row <strong>LI</strong> and column <strong>LJ</strong> of the unit: if <strong>NI</strong> and <strong>NJ</strong> are absent, zero or unity, a single element (or pair of elements) is defined; if <strong>NI</strong> and <strong>NJ</strong> are both positive, then <strong>NIxNJ</strong> elements (or pairs of elements) are identified in a double FORTRAN-like loop, starting with the element (or pair of elements) at (<strong>LI,LJ</strong>).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ID</strong></td>
<td>spring stiffness-displacement table identifier (see record I–4a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>NI,NJ</strong></td>
<td>element-identification looping parameters, as discussed in <strong>LI,LJ</strong>, above</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NSRF** (T-820) surface-method parameter

**NPTS** (T-820) point-method parameter

if ( **NSRF** contact-surface elements have been identified ) then
if ( **NPTS** > 0 ) then go to T-820c
else go to T-820d
else return to T-820a
**T-820b E820 Element-Number Contact-Element Specifications**

NSRF < 0 (T-820) indicates that the element-number method is to be used to identify the contiguous shell elements that comprise the contact surface. This method can (but generally should not) be used to identify contact elements that are associated with shell units, and it must be used to identify elements that are defined in STAGS element units (see Note 1 for the T-820 record). With this method, the magnitude of the NSRF parameter indicates the minimum number of contact elements to be identified with the set of one or more T-820b records included for the current contact-definition. The number of contact elements identified by any given T-820b depends on the parameters on that record.

<table>
<thead>
<tr>
<th>USRF</th>
<th>TYPE</th>
<th>I1</th>
<th>I2</th>
<th>INC</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>USRF</td>
<td>identifies the shell or element unit within which the element(s) identified by the I1, I2 and INC parameters are defined</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TYPE</td>
<td>specifies the element type for this (or these) element(s): TYPE must currently be either E410 or E320</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I1, I2, INC</td>
<td>element identifiers: I1 &gt; 0, with I2 = 0 and INC = 0 identifies element # I1 (in unit USRF) as part of the contact surface for the current contact-definition; I1, I2, INC &gt; 0 identifies I2 elements—starting with element I1 and incrementing by INC until I2 elements have been indicated (in unit USRF) as part of the contact surface for the current contact-definition</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>spring stiffness-displacement table identifier (see record I–4a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NSRF (T-820) surface-method parameter
NPTS (T-820) point-method parameter

if (fewer than NSRF contact-surface elements have been identified) then
  return to T-820b
elseif (NPTS > 0) then
  go to T-820c
else
  go to T-820d
**T-820c Row & Column Contact-Point Specifications**

**NPTS** ≥ 0 (T-820) indicates that the row & column method is to be used to identify the points that may come into contact with the contact surface specified for the current contact-definition. This method clearly can only be used to identify nodes that are associated with one or more shell units, because there are no row or column associations for nodal points in element units. With this method, the value of the NPTS parameter indicates the minimum number of contact points to be identified with the set of one or more T-820c records included for the current contact-definition (see Note 2 for T-820). The number of contact points identified by any given T-820c depends on the parameters on that record.

<table>
<thead>
<tr>
<th>UNITP</th>
<th>LI</th>
<th>LJ</th>
<th>RADIUS</th>
<th>TOUCHE</th>
<th>NI</th>
<th>NJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNITP</td>
<td>identifies the shell unit within which the contact point(s) identified by the LI and LJ row &amp; column parameters are defined; the nodal mesh for this shell unit has NROWS rows and NCOLS columns of nodal points</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LI,LJ</td>
<td>row and column numbers identifying one or more contact points: LI = LJ = 0 indicates that all of the node points of shell unit UNITP are to be considered as contact points; LI &gt; 0 with LJ = 0 identifies all of the nodes in row LI of the nodal grid; LJ &gt; 0 with LI = 0 identifies all of the nodes in column LJ; and LI &gt; 0 with LJ &gt; 0 identifies one or more nodes, starting at row LI and column LJ of the unit: if NI and NJ are absent, zero or unity, a single point (LI,LJ) is defined; if NI and NJ are both positive, then NIXNJ points are identified in a double FORTRAN-like loop, starting with the point at (LI,LJ).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RADIUS</td>
<td>specifies a radial offset parameter to be used in determining whether or not the point(s) identified by the current T-820c are in contact with the contact surface: this parameter can be used to account for the thickness of material surrounding the contact point(s) in much the same way as the thicknesses of contact elements on the contact surface are taken into account</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOUCHE</td>
<td>provides information required if the identified contact point(s) are in contact with the contact surface at the outset of the problem: set TOUCHE = 0 if not; or set TOUCHE = 1 if the point(s) are contacting the surface from its positive side; or set TOUCHE = -1 if the point(s) are contacting the surface from its negative side—where the normal vector for the contact surface points out of the positive side of the surface and away from the negative side</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NI,NJ</td>
<td>point-identification looping parameters, as discussed in LI,LJ, above</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

if ( fewer than NPTS contact-points have been identified ) then
  go to T-820c
else
  return to T-100
T-820d Point-Number Contact-Point Specifications

NPTS < 0 (on T-820) indicates that the point-number method is to be used to identify the nodal points that may come into contact with the contact surface. This method can (but generally should not) be used to identify nodes that are associated with shell units, and it must be used to identify points that are defined in STAGS element units. With this method, the magnitude of the NPTS parameter indicates the minimum number of contact points to be identified with the set of one or more T-820d records included for the current contact-definition (see Note 2 for the T-820 record). The number of contact points identified by any given T-820d depends on the parameters on that record.

UNITP  I1  I2  INC  RADIUS  TOUCHE

UNITP identifies the shell or element unit within which the contact point(s) identified by the I1, I2 and INC parameters are defined; the nodal mesh for this shell unit has NROWS rows and NCOLS columns of node points.

I1,I2,INC point identifiers: I1 > 0, with I2 = INC = 0 identifies point # I1 (in unit UNITP) as a contact point for the current contact-definition; I1, I2, INC > 0 identifies I2 points—starting with point I1 and incrementing by INC until I2 points have been indicated (in unit UNITP) as contact points for the current contact-definition.

RADIUS specifies a radial offset parameter to be used in determining whether or not the point(s) identified by the current T-820d are in contact with the contact surface: this parameter can be used to account for the thickness of material surrounding the contact point(s) in much the same way as the thicknesses of contact elements on the contact surface are taken into account.

TOUCHE provides information required if the identified contact point(s) are in contact with the contact surface at the outset of the problem: set TOUCHE = 0 if not; or set TOUCHE = 1 if the point(s) are contacting the surface from its positive side; or set TOUCHE = −1 if the point(s) are contacting the surface from its negative side—where the normal vector for the contact surface points out of the positive side of the surface and away from the negative side.

if (fewer than NPTS contact-points have been identified) then
  return to T-820d
else
  return to T-100
T-822 Line-Contact Interaction Definition

The T-822 record specifies a pair of “contact” lines (which must have been defined via S-5 records, as described on page 7-17) that may experience line-to-line contact with each other during the course of the analysis that STAGS is to perform. The first line of each such pair must have one or more type E410 elements associated with it (qualifying it to be a contacted line), as described earlier. The second line, which may but does not need to have elements associated with it, is treated as the contacting line. When contact occurs at one or more points along the two lines, STAGS uses penalty information supplied by the analyst to generate stiffnesses and forces on the fly to treat that contact.

<table>
<thead>
<tr>
<th>LINE1</th>
<th>LINE2</th>
<th>IPEN</th>
<th>NX</th>
<th>INC1</th>
<th>INC2</th>
<th>INC3</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINE1</td>
<td>LINE2</td>
<td>IPEN</td>
<td>NX</td>
<td>INC1</td>
<td>INC2</td>
<td>INC3</td>
</tr>
</tbody>
</table>

LINE1 identifies the contacted line, which must have one or more elements associated with it; this line must have been specified via an S-5 record, as described on page 7-17.

LINE2 identifies the contacting line, which may (but is not required to) have one or more elements associated with it; this line must also have been specified via an S-5 record.

IPEN identifies the penalty function table (of stiffness as a function of penetration) that STAGS is to use in calculating stiffnesses and forces that arise when contact occurs between these two lines; the table identified must have been specified via an I-4 record set, as described above.

NX indicates the number of line contact interactions that are to be specified with information on this T-822 record; STAGS sets NX = 1 if it is not positive.

INC1 incrementation parameter for LINE1 (used only when NX > 0)

INC2 incrementation parameter for LINE2 (used only when NX > 0)

INC3 incrementation parameter for IPEN (used only when NX > 0)

return to T-100
9.6 Definition of Sandwich Elements via the Ecom Protocol

Sandwich structures play such an important role in the design of many aerospace structures, so it is necessary that their behavior be determined adequately. The classical approach to the modeling of sandwiches is an extension of thin-shell-theory, in which the behavior of a three-dimensional sandwich object is reduced to the behavior of a two-dimensional surface with in- and out-of-plane stiffness properties. Here, the sandwich construction is idealized as a pair of membranes that are held apart by a core that has a relatively large resistance against transverse shear. The core is virtually inextensional in the transverse direction, with virtually no stiffness in the middle-surface plane. This type of model is called a sandwich of the first kind. Finite element discretizations of this kind of classical model have been made in the past, but successful applications have been confined primarily to linear or moderately nonlinear analyses. A more general sandwich model emerges by considering the two faces to be shells (with bending as well as the usual membrane stiffness) that are held apart by a lightweight core. The core may have three dimensional elastic properties. This type of model is called a sandwich of the second kind, and is the model that is implemented in STAGS.* A good example of a problem requiring this model would be one with structural walls made of glass fiber faces and using polyurethane foam as the core material.

There are five sandwich elements in the current version of STAGS. Three of these (E830, E840 and E849) are “standard” 6-, 8- and 18-node 3-layered (shell/solid/shell) counterparts of the E330, E410 and E480 thin shell elements, respectively. The other two (E845 and E847) are 10- and 14-node 3-layered counterparts of the E510 and E710 shell elements—most appropriately used in modeling transition zones where E840 mesh refinements are required.

The 6-node E830 element is a “standard” sandwich element that is constructed with a pair of E330 triangular thin shell elements that are held apart by a 6-node pentahedral solid core component. One or more E830 elements (or additional E830 elements) can be defined with the T-830 record set. For more information about the E830 element, see “E830 6–Node sandwich element” on page 14-36.

The 8-node E840 element is a “standard” sandwich element that is constructed with a pair of 4-node E410 quadrilateral thin shell elements that are held apart by an 8-node hexahedral solid

* See, for example:
core component. One or more E840 elements (or additional E840 elements) can be defined with the T-840 record set. For more information about the E840 element, see “E840 8–Node sandwich element” on page 14-44.

The 10-node E845 element is the sandwich-element counterpart of the E510 thin-shell mesh-transition “quadrilateral” element. An E845 element is constructed with a pair of 5-node E510 transition elements that are held apart by a 10-node solid-core mesh-transition component. One or more E845 elements (or additional E845 elements) can be defined with the T-845 record set. For more information about the E845 element, see “E845 and E847 mesh-transition sandwich elements” on page 14-46.

The 14-node E847 element is the sandwich-element counterpart of the E710 thin-shell mesh-transition “quadrilateral” element. An E847 element is constructed with a pair of 7-node E710 transition elements that are held apart by a 14-node solid-core mesh-transition component. One or more E847 elements (or additional E847 elements) can be defined with the T-847 record set. For more information about the E847 element, see “E845 and E847 mesh-transition sandwich elements” on page 14-46.

The 18-node E849 element is a “standard” sandwich element that is constructed with a pair of 9-node E480 quadrilateral thin shell elements that are held apart by an 18-node hexahedral solid core component. One or more E849 elements (or additional E849 elements) can be defined with the T-849 record set. For more information about the E849 element, see “E849 18–Node sandwich element” on page 14-45.
T-830 E830 6-Node Sandwich Element Definition

The E830 6-node sandwich element in STAGS is constructed with a pair of E330 triangular shells (with bending as well as the usual membrane stiffness) that are held apart by a lightweight core that has three-dimensional elastic properties.

Figure 9.3 E830 6-Node Sandwich Element

Definition of one or more E830 6-node sandwich elements is accomplished with the following set of records:

T-830 to specify parameters used for all parts of the element
T-830a to specify information for the lower face sheet of the element
T-830b to specify lower face sheet transformation angles (optional)
T-830c to specify information for the upper face sheet
T-830d to specify upper face sheet transformation angles (optional)
T-830e to specify core parameters
T-830f to specify core transformation angles (optional)
T-830g for “x-direction” incrementation parameters
T-830h for “y-direction” incrementation parameters

These are described sequentially in the following text.
<table>
<thead>
<tr>
<th>KELT</th>
<th>element code number (must be 830)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILIN</td>
<td>geometric-linearity flag:</td>
</tr>
<tr>
<td></td>
<td>0  –  nonlinear strain-displacement relations</td>
</tr>
<tr>
<td></td>
<td>1  –  linear strain-displacement relations</td>
</tr>
<tr>
<td>INTEG</td>
<td>number of surface-integration points: set INTEG = 1, 3, 4 or 7; if INTEG = 0, STAGS sets INTEG = 3</td>
</tr>
<tr>
<td>IPEN</td>
<td>penalty option (see N-1)</td>
</tr>
<tr>
<td></td>
<td>0  –  no penalty function on fourth-order terms in E330 elements</td>
</tr>
<tr>
<td></td>
<td>1  –  penalty function included in E330 elements</td>
</tr>
<tr>
<td>NX</td>
<td>x-direction looping parameter, set equal to unity by STAGS if omitted or nonpositive; NX &gt; 1 instructs STAGS to generate a set of NX type E830 elements in the x direction, using nodal incrementation information given on T-830g</td>
</tr>
<tr>
<td>NY</td>
<td>y-direction looping parameter, set equal to unity by STAGS if omitted or nonpositive; NY &gt; 1 instructs STAGS to generate a set of NY type E830 elements in the y direction for each of the NX elements generated in the x direction in the current definition, using nodal incrementation information given on T-830h</td>
</tr>
<tr>
<td>USERC</td>
<td>user-specified element number for core component</td>
</tr>
<tr>
<td></td>
<td>(used only if IUWLE = 1 on H-1)</td>
</tr>
<tr>
<td>USER1</td>
<td>user-specified element number for lower face sheet, (if IUWLE = 1 on H-1)</td>
</tr>
<tr>
<td>USER2</td>
<td>user-specified element number for upper face sheet, (if IUWLE = 1 on H-1)</td>
</tr>
</tbody>
</table>

*go to T-830a*
**T-830a E830 Lower Face-Sheet Properties**

A single T-830a record must be included immediately following the T-830 record, to specify the nodes for the lower face sheet of the first E830 element and to specify other parameters for the lower face sheet of each of the element(s) specified in the current definition.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>IFABL</th>
<th>ZETAL</th>
<th>ECZL</th>
<th>IPLASL</th>
<th>IANGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>first node point on lower face sheet</td>
<td>second node point on lower face sheet</td>
<td>third node point on lower face sheet</td>
<td>fabrication identifier for the lower face sheet element(s); this is equivalent to the WALL parameter for E330 triangular elements:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;0 – wall configuration number in the Wall Fabrication Table (K-1)</td>
<td>0 – shell wall properties are given in user-written subroutine WALL (See the T-4 record description for more information about this)</td>
<td>&lt;0 – shell fabrication identifier in the GCP Fabrication Table (I-21a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>angle $\zeta$ between the wall-reference coordinate $x_w$ and the fabrication coordinate $\bar{x}$, for the lower face sheet; $\zeta$ is a right-handed rotation about $\bar{z}$; see Figure 8.2 on page 8-19.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>eccentricity in $z'$ direction, for the lower face sheet; ECZL is the $z'$ coordinate of the shell wall middle surface; please refer to Figure 6.2 on page 6-28; in element units, $(x', y', z')$ are used in place of $(X', Y', Z')$, which do not exist in an element unit; see “Effects of Eccentricity” on page 16-6.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lower face sheet material-nonlinearity flag:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 – elastic behavior only</td>
<td>1 – plasticity included, with the material law satisfied at each element integration point</td>
<td>2 – plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**IANGL** lower face sheet wall-reference option; see the discussion on page 8-18

0 - use default strategy of projecting \( x_g, y_g \) to establish \( x_w \)

1 - input \( r_w \), which is projected to establish \( x_w \); see (T-830b)

\[
\text{if ( IANGL > 0 ) then} \quad \text{go to T-830b}
\]

\[
\text{else} \quad \text{go to T-830c}
\]
**T-830b E830 Lower Face-Sheet Wall Reference Vector**

**RXL RYL RZL**

RXL, RYL, RZL lower face sheet wall reference vector, \( \mathbf{r}_w \), which is projected onto the element surface to determine the direction of the wall-reference coordinate \( x_w \); \( \mathbf{r}_w \) is expressed in \((x_g, y_g, z_g)\) global coordinates.

*go to* T-830c
**T-830c E830 Upper Face-Sheet Properties**

A single T-830c record must be included immediately following the T-830a (or T-830b) record, to specify the nodes for the upper face sheet of the first E830 element and to specify other parameters for the upper face sheet of each of the element(s) specified in the current definition.

<table>
<thead>
<tr>
<th>N4</th>
<th>N5</th>
<th>N6</th>
<th>IFABU</th>
<th>ZETAU</th>
<th>ECZU</th>
<th>IPLASU</th>
<th>IANGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>N4</td>
<td>first node point on upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N5</td>
<td>second node point on upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N6</td>
<td>third node point on upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFABU</td>
<td>fabrication identifier for the upper face sheet element(s); this is equivalent to the WALL parameter for E330 quadrilateral elements:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;0</td>
<td>– wall configuration number in the Wall Fabrication Table (K-1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>– shell wall properties are given in user-written subroutine WALL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;0</td>
<td>– shell fabrication identifier in the GCP Fabrication Table (I-21a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZETAU</td>
<td>angle $\zeta$ between the wall-reference coordinate $x_w$ and the fabrication coordinate $x$, for the upper face sheet; $\zeta$ is a right-handed rotation about $\zeta$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECZU</td>
<td>eccentricity in $z'$ direction, for the upper face sheet; ECZL is the $z'$ coordinate of the shell wall middle surface</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPLASU</td>
<td>upper face sheet material-nonlinearity flag:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>– elastic behavior only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>– plasticity included, with the material law satisfied at each element integration point</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>– plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IANGL</td>
<td>upper face sheet wall-reference option; see discussion on page 8-18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>– use default strategy of projecting $x_w, y_g$ to establish $x_w$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>– input $r_w$, which is projected to establish $x_w$; see (T-830d)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
if ( IANGU > 0 ) then go to T-830d
else go to T-830e
```
**T-830d E830 Upper Face-Sheet Wall Reference Vector**

**RXU, RYU, RZU**

upper face sheet wall reference vector, \( \mathbf{r}_w \), which is projected onto the element surface to determine the direction of the wall-reference coordinate \( x_w \); \( \mathbf{r}_w \) is expressed in \((x_g, y_g, z_g)\) global coordinates

*go to T-830e*
T-830e E830 Core Properties

A single T-830e record must be included immediately following the T-830c (or T-830d) record, to specify parameters for the core component of the element(s) specified in the current definition.

IFABC  ZETAC  IPLASC  IANGC

IFABC  
core fabrication identifier:
>0  –  wall configuration number in the Wall Fabrication Table (K-1)
<0  –  solid fabrication identifier in the GCP Fabrication Table (I-22a)

ZETAC  
angle $\zeta$ between the wall-reference coordinate $x_w$ and the fabrication coordinate $x$, for the core component; $\zeta$ is a right-handed rotation about $z$

IPLASC  
core material-nonlinearity flag:
0  –  elastic behavior only
1  –  plasticity included (not operational with GCP fabrications, yet)

IANGC  
core wall-reference option; see discussion on page 8-18
0  –  use default strategy of projecting $x'_w$ to establish $x_w$
1  –  input $r_w$, which is projected to establish $x_w$; see (T-830f)

NX  (T-830)  x-direction looping parameter
NY  (T-830)  y-direction looping parameter

if  ( IANGC > 0 ) then  go to T-830f
elseif ( NX > 0 ) then  go to T-830g
elseif ( NY > 0 ) then  go to T-830h
else  return to T-100


T-830f E830 Core Reference Vector

RXC, RYC, RZC core reference vector, $r_w$, which is projected onto the element surface to determine the direction of the core-reference coordinate $x_w$; $r_w$ is expressed in $(x_g, y_g, z_g)$ global coordinates.

- **NX** (T-830) $x$-direction looping parameter
- **NY** (T-830) $y$-direction looping parameter

if $(NX > 0)$ then go to T-830g
elseif $(NY > 0)$ then go to T-830h
else return to T-100
**T-830g X-Direction Incrementations**

A single record of type T-830g must be included immediately after the T-830e or T-830f record when the **nx** “x-direction” **looping parameter** (T-830) is greater than unity. Eight nodal incrementation variables are specified here for use with the “x-direction” looping function on the T-830 record. **nx E830** sandwich elements are generated in the “x-direction” with this information. Three incrementation variables are also specified here for “x-direction” looping with the user-element-number parameters on T-830.

<table>
<thead>
<tr>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>I4</th>
<th>I5</th>
<th>I6</th>
<th>I7</th>
<th>I8</th>
<th>I9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>incrementation for the <strong>N1</strong> lower face sheet node on the T-830a record</td>
<td>incrementation for the <strong>N2</strong> lower face sheet node on the T-830a record</td>
<td>incrementation for the <strong>N3</strong> lower face sheet node on the T-830a record</td>
<td>incrementation for the <strong>N4</strong> upper face sheet node on the T-830c record</td>
<td>incrementation for the <strong>N5</strong> upper face sheet node on the T-830c record</td>
<td>incrementation for <strong>USERC</strong> on the T-830 record</td>
<td>incrementation for <strong>USER1</strong> on the T-830 record</td>
<td>incrementation for <strong>USER2</strong> on the T-830 record</td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero, or positive, as required.

**Example:** the following T-830 / T-830a / T-830c / T-830g record combination:

```
830 1 0 0 3 $ T-830 with NX=3
10 20 30 ... $ T-830a lower face nodes
50 60 70 ... $ T-830c upper face nodes
...
1 1 1 2 2 2 $ T-830g increments
```

generates three **E830** elements, with the following nodes:

```
10 20 30 50 60 70 Element #1
11 21 31 52 62 72 Element #2
12 22 32 54 64 74 Element #3
```

- **NY** (T-830)  y-direction looping parameter
  - if **( NY > 0 )** then *go to T-830h*
  - else *return to T-100*
T-830h Y-Direction Incrementations

A single type T-830h record must be included immediately after the T-830e or T-830f or T-830g record when the NY “y-direction” looping parameter is greater than unity (T-830). Eight nodal incrementation variables are specified here for use with the T-830 record “y-direction” looping function. NY E830 sandwich elements are generated in the “y-direction”, for each E830 element generated in the x direction, with this information. The NX and NY looping parameters are usually employed together to specify NX × NY E830 elements in a single stroke.

<table>
<thead>
<tr>
<th>J1</th>
<th>J2</th>
<th>J3</th>
<th>J4</th>
<th>J5</th>
<th>J6</th>
<th>J7</th>
<th>J8</th>
<th>J9</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1</td>
<td>incrementation for the N1 lower face sheet node on the T-830a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J2</td>
<td>incrementation for the N2 lower face sheet node on the T-830a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J3</td>
<td>incrementation for the N3 lower face sheet node on the T-830a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J4</td>
<td>incrementation for the N4 upper face sheet node on the T-830c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J5</td>
<td>incrementation for the N5 upper face sheet node on the T-830c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J6</td>
<td>incrementation for the N6 upper face sheet node on the T-830c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J7</td>
<td>incrementation for USERC on the T-830 record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J8</td>
<td>incrementation for USER1 on the T-830 record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J9</td>
<td>incrementation for USER2 on the T-830 record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero, or positive, as required.

Example: the following T-830 / T-830a / T-830c / T-830g / T-830h record combination:

```
830 1 0 0 3 2
10 20 30 ...
50 60 70 ...
...
1 1 1 1 1 1
5 5 5 5 5
```

$ T-830 NX=3, NY=2
$ T-830a (lower face nodes)
$ T-830c (upper face nodes)
$ T-830g (x increments)
$ T-830h (y increments)

generates six E830 elements, with the following nodes:

```
10 20 30 50 60 70  Element #1
11 21 31 51 61 71  Element #2
12 22 32 52 62 72  Element #3
15 25 35 55 65 75  Element #4
16 26 36 56 66 76  Element #5
17 27 37 57 67 77  Element #6
```

return to T-100
**T-840 E840 8-Node Sandwich Element Definition**

The **E840** 8-node sandwich element in **STAGS**, shown in Figure 9.4, is constructed with a pair of **E410** quadrilateral shells (with bending as well as the usual membrane stiffness) that are held apart by a lightweight core that has three-dimensional elastic properties.

![Diagram of E840 8-Node Sandwich Element](image)

**Figure 9.4** **E840** 8-Node Sandwich Element

Definition of one or more **E840** 8-node sandwich elements is accomplished with the following set of records:

- **T-840**: to specify parameters used for all parts of the element
- **T-840a**: to specify information for the lower face sheet of the element
- **T-840b**: to specify lower face sheet transformation angles (optional)
- **T-840c**: to specify information for the upper face sheet
- **T-840d**: to specify upper face sheet transformation angles (optional)
- **T-840e**: to specify core parameters
- **T-840f**: to specify core transformation angles (optional)
- **T-840g**: for “x-direction” incrementation parameters
- **T-840h**: for “y-direction” incrementation parameters

These are described sequentially in the following text.
KELT  ILIN  INTEG  IPEN  NX  NY  USERC USER1 USER2

KELT  

Element code number (must be 840)

ILIN  

Geometric-linearity flag:

0 – nonlinear strain-displacement relations
1 – linear strain-displacement relations

INTEG  

Integration-type flag (see N-1):

0 – standard integration, or 2 × 2 Gauss points for E410 elements
1 – modified 5-point integration, previously referred to as full integration

IPEN  

Penalty option (see N-1)

0 – no penalty function on fourth-order terms in E410 elements
1 – penalty function included in E410 elements

NX  

X-direction looping parameter, set equal to unity by STAGS if omitted or nonpositive; NX > 1 instructs STAGS to generate a set of NX type E840 elements in the x direction, using nodal incrementation information given on T-840g

NY  

Y-direction looping parameter, set equal to unity by STAGS if omitted or nonpositive; NY > 1 instructs STAGS to generate a set of NY type E840 elements in the y direction for each of the NX elements generated in the x direction in the current definition, using nodal incrementation information given on T-840h

USERC  

User-specified element number for core component

(used only if IUWLE = 1 on H-1)

USER1  

User-specified element number for lower face sheet, (if IUWLE = 1 on H-1)

USER2  

User-specified element number for upper face sheet, (if IUWLE = 1 on H-1)

* go to T-840a
**T-840a E840 Lower Face-Sheet Properties**

A single T-840a record must be included immediately following the T-840 record, to specify the nodes for the lower face sheet of the first E840 element and to specify other parameters for the lower face sheet of each of the element(s) specified in the current definition.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
<th>IFABL</th>
<th>ZETAL</th>
<th>ECZL</th>
<th>IPLASL</th>
<th>IANGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>first node point on lower face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>second node point on lower face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>third node point on lower face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N4</td>
<td>fourth node point on lower face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFABL</td>
<td>fabrication identifier for the lower face sheet element(s); this is equivalent to the WALL parameter for E410 quadrilateral elements:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;0</td>
<td>wall configuration number in the Wall Fabrication Table (K-1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>shell wall properties are given in user-written subroutine WALL (See the T-4 record description for more information about this)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;0</td>
<td>shell fabrication identifier in the GCP Fabrication Table (I-21a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZETAL</td>
<td>angle $\zeta$ between the wall-reference coordinate $x_w$ and the fabrication coordinate $\bar{x}$, for the lower face sheet; $\zeta$ is a right-handed rotation about $\bar{z}$; see Figure 8.2 on page 8-19.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECZL</td>
<td>eccentricity in $z'$ direction, for the lower face sheet; ECZL is the $z'$ coordinate of the shell wall middle surface; please refer to Figure 6.2 on page 6-28; in element units, $(x', y', z')$ are used in place of $(X', Y', Z')$, which do not exist in an element unit; see “Effects of Eccentricity” on page 16-6.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPLASL</td>
<td>lower face sheet material-nonlinearity flag:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>elastic behavior only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>plasticity included, with the material law satisfied at each element integration point</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
IANGL lower face sheet wall-reference option; see discussion on page 8-18

0  – use default strategy of projecting $x_g, y_g$ to establish $x_w$
1  – input $r_w$, which is projected to establish $x_w$; see (T-840b)

if ( IANGL > 0 ) then
  go to T-840b
else
  go to T-840c
T-840b E840 Lower Face-Sheet Wall Reference Vector

RXL, RYL, RZL

lower face sheet wall reference vector, \( \mathbf{r}_w \), which is projected onto the element surface to determine the direction of the wall-reference coordinate \( x_w \); \( \mathbf{r}_w \) is expressed in global coordinates \( (x_g, y_g, z_g) \)

\( \text{go to T-840c} \)
**T-840c E840 Upper Face-Sheet Properties**

A single T-840c record must be included immediately following the T-840a (or T-840b) record, to specify the nodes for the upper face sheet of the first E840 element and to specify other parameters for the upper face sheet of each of the element(s) specified in the current definition.

<table>
<thead>
<tr>
<th>N5</th>
<th>N6</th>
<th>N7</th>
<th>N8</th>
<th>IFABU</th>
<th>ZETAU</th>
<th>ECZU</th>
<th>IPLASU</th>
<th>IANGU</th>
</tr>
</thead>
<tbody>
<tr>
<td>first node point on upper face sheet</td>
<td>second node point on upper face sheet</td>
<td>third node point on upper face sheet</td>
<td>fourth node point on upper face sheet</td>
<td>fabrication identifier for the upper face sheet element(s); this is equivalent to the WALL parameter for E410 quadrilateral elements:</td>
<td>angle ζ between the wall-reference coordinate ( x_w ) and the fabrication coordinate ( \hat{x} ), for the upper face sheet; ( \zeta ) is a right-handed rotation about ( \zeta )</td>
<td>eccentricity in ( \zeta' ) direction, for the upper face sheet; ECZL is the ( \zeta' ) coordinate of the shell wall middle surface</td>
<td>upper face sheet material-nonlinearity flag:</td>
<td>upper face sheet wall-reference option; see discussion on page 8-18</td>
</tr>
<tr>
<td>&gt;0 – wall configuration number in the Wall Fabrication Table (K-1)</td>
<td>0 – shell wall properties are given in user-written subroutine WALL</td>
<td>&lt;0 – shell fabrication identifier in the GCP Fabrication Table (I-21a)</td>
<td>if ( IANGU &gt; 0 ) then ( \text{go to T-840d} )</td>
<td>0 – elastic behavior only</td>
<td>1 – plasticity included, with the material law satisfied at each element integration point</td>
<td>2 – plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>else ( \text{go to T-840e} )</td>
<td>0 – use default strategy of projecting ( x_{g'} y_g ) to establish ( x_w )</td>
<td>1 – input ( r_w ), which is projected to establish ( x_w ); see (T-840d)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**T-840d E840 Upper Face-Sheet Wall Reference Vector**

RXU, RYU, RZU upper face sheet wall reference vector, \( \mathbf{r}_w \), which is projected onto the element surface to determine the direction of the wall-reference coordinate \( x_w \); \( \mathbf{r}_w \) is expressed in \((x_g,y_g,z_g)\) global coordinates.

go to **T-840e**
### T-840e E840 Core Properties

A single T-840e record must be included immediately following the T-840c (or T-840d) record, to specify parameters for the core component of the element(s) specified in the current definition.

#### IFABC  ZETAC  IPLASC  IANGC

**IFABC**

core fabrication identifier:

- \( >0 \) – wall configuration number in the Wall Fabrication Table (K-1)
- \( <0 \) – solid fabrication identifier in the GCP Fabrication Table (I-22a)

**ZETAC**

angle \( \zeta \) between the wall-reference coordinate \( x_w \) and the fabrication coordinate \( x \), for the core component; \( \zeta \) is a right-handed rotation about \( z \)

**IPLASC**

core material-nonlinearity flag:

- \( 0 \) – elastic behavior only
- \( 1 \) – plasticity included (not operational with GCP fabrications, yet)

**IANGC**

core wall-reference option; see discussion on page 8-18

- \( 0 \) – use default strategy of projecting \( x_{e'}y_{g} \) to establish \( x_w \)
- \( 1 \) – input \( r_w \), which is projected to establish \( x_w \); see (T-840f)

**NX** (T-840) x-direction looping parameter

**NY** (T-840) y-direction looping parameter

if \( ( \text{IANGC} > 0 ) \) then go to T-840f

elseif \( ( \text{NX} > 0 ) \) then go to T-840g

elseif \( ( \text{NY} > 0 ) \) then go to T-840h

else return to T-100
**T-840f E840 Core Reference Vector**

**RXC, RYC, RZC**

RXC, RYC, RZC core reference vector, \( \mathbf{r}_w \), which is projected onto the element surface to determine the direction of the core-reference coordinate \( x_w \); \( \mathbf{r}_w \) is expressed in \((x_g, y_g, z_g)\) global coordinates.

- **NX** \((T-840)\): x-direction looping parameter
- **NY** \((T-840)\): y-direction looping parameter

If \( \text{NX} > 0 \) then *go to T-840g*

Elseif \( \text{NY} > 0 \) then *go to T-840h*

Else return to *T-840*
**T-840g X-Direction Incrementations**

A single record of type T-840g must be included immediately after the T-840e or T-840f record when the **nx** “x-direction” **looping parameter** (T-840) is greater than unity. Eight nodal incrementation variables are specified here for use with the “x-direction” looping function on the T-840 record. **nx E840** sandwich elements are generated in the “x-direction” with this information. Three incrementation variables are also specified here for “x-direction” looping with the user-element-number parameters on T-840.

<table>
<thead>
<tr>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>I4</th>
<th>I5</th>
<th>I6</th>
<th>I7</th>
<th>I8</th>
<th>I9</th>
<th>I10</th>
<th>I11</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **I1** incrementation for the **N1** lower face sheet node on the T-840a record
- **I2** incrementation for the **N2** lower face sheet node on the T-840a record
- **I3** incrementation for the **N3** lower face sheet node on the T-840a record
- **I4** incrementation for the **N4** lower face sheet node on the T-840a record
- **I5** incrementation for the **N5** upper face sheet node on the T-840c record
- **I6** incrementation for the **N6** upper face sheet node on the T-840c record
- **I7** incrementation for the **N7** upper face sheet node on the T-840c record
- **I8** incrementation for the **N8** upper face sheet node on the T-840c record
- **I9** incrementation for **USERC** on the T-840 record
- **I10** incrementation for **USER1** on the T-840 record
- **I11** incrementation for **USER2** on the T-840 record

Any of these incrementation variables can be negative, zero, or positive, as required.

**Example:** the following T-840 / T-840a / T-840c / T-840g record combination:

```
840 1 0 0 3 $ T-840 with NX=3
10 20 30 40 ... $ T-840a lower face nodes
50 60 70 80 ... $ T-840c upper face nodes
... 1 1 1 1 2 2 2 2 $ T-840g increments
```

generates three **E840** elements, with the following nodes:

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>21</td>
<td>31</td>
<td>41</td>
<td>52</td>
<td>62</td>
<td>72</td>
<td>82</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>22</td>
<td>32</td>
<td>42</td>
<td>54</td>
<td>64</td>
<td>74</td>
<td>84</td>
<td></td>
</tr>
</tbody>
</table>

| Element #1 | Element #2 | Element #3 |

---

**NY** (T-840)  **y-direction looping parameter**

if **NY > 0** then **go to T-840h**

else **return to T-100**
T-840h Y-Direction Incrementations

A single type T-840h record must be included immediately after the T-840e or T-840f or T-840g record when the NY “y-direction” looping parameter is greater than unity (T-840). Eight nodal incrementation variables are specified here for use with the T-840 record “y-direction” looping function. NY E840 sandwich elements are generated in the y direction, for each E840 element generated in the x direction, with this information. The NX and NY looping parameters are usually employed together to specify NX × NY E840 elements in a single stroke.

J1 J2 J3 J4   J5 J6 J7 J8   J9 J10 J11

J1 incrementation for the N1 lower face sheet node on the T-840a record
J2 incrementation for the N2 lower face sheet node on the T-840a record
J3 incrementation for the N3 lower face sheet node on the T-840a record
J4 incrementation for the N4 lower face sheet node on the T-840a record
J5 incrementation for the N5 upper face sheet node on the T-840c record
J6 incrementation for the N6 upper face sheet node on the T-840c record
J7 incrementation for the N7 upper face sheet node on the T-840c record
J8 incrementation for the N8 upper face sheet node on the T-840c record
J9 incrementation for USERC on the T-840 record
J10 incrementation for USER1 on the T-840 record
J11 incrementation for USER2 on the T-840 record

Any of these incrementation variables can be negative, zero, or positive, as required.

Example: the following T-840 / T-840a / T-840c / T-840g / T-840h record combination:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>840</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

$ T-840$ NX=3, NY=2
$ T-840a$ (lower face nodes)
$ T-840c$ (upper face nodes)
$ T-840g$ (x increments)
$ T-840h$ (y increments)

generates six E840 elements, with the following nodes:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>60</td>
<td>70</td>
<td>80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>21</td>
<td>31</td>
<td>41</td>
<td>51</td>
</tr>
<tr>
<td>61</td>
<td>71</td>
<td>81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>22</td>
<td>32</td>
<td>42</td>
<td>52</td>
</tr>
<tr>
<td>62</td>
<td>72</td>
<td>82</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>23</td>
<td>33</td>
<td>43</td>
<td>53</td>
</tr>
<tr>
<td>63</td>
<td>73</td>
<td>83</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>24</td>
<td>34</td>
<td>44</td>
<td>54</td>
</tr>
<tr>
<td>64</td>
<td>74</td>
<td>84</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Element #1
Element #2
Element #3
Element #4
Element #5
Element #6

return to T-100
**T-845 E845 10-Node Sandwich Transition Element Definition**

The **E845** 10-node sandwich-transition element, shown in Figure 9.5, is used in sandwich shell units to make a transition between one mesh and another that has nodes that are exactly half as far apart as the one in which the **E845** element “resides.” It can be used to accomplish the same objective in an element unit. A single T-845 definition may specify one or more **E845** sandwich transition elements.

![Figure 9.5 E845 10-Node Sandwich Transition Elements](image)

Definition of one or more **E845** 10-node sandwich transition elements is accomplished with the following set of records:

- **T-845** to specify parameters used for all parts of the element
- **T-845a** to specify information for the lower face sheet of the element
- **T-845b** to specify lower face sheet transformation angles (optional)
- **T-845c** to specify information for the upper face sheet
- **T-845d** to specify upper face sheet transformation angles (optional)
- **T-845e** to specify core parameters
- **T-845f** to specify core transformation angles (optional)
- **T-845g** for “x-direction” incrementation parameters
- **T-845h** for “y-direction” incrementation parameters

These are described sequentially in the following text.
KELT element code number (must be 845)
ILIN geometric-linearity flag:
   0 – nonlinear strain-displacement relations
   1 – linear strain-displacement relations
INTEG integration-type flag (see N-1):
   0 – standard integration, or 2×2 Gauss points for E410 elements
   1 – modified 5-point integration, previously referred to as full integration
IPEN penalty option (see N-1)
   0 – no penalty function on fourth-order terms in E410 elements
   1 – penalty function included in E410 elements
IEDGE identifies the E845 edge on which node N5 is located (N10 is above N5):
   1 – node N5 is between N3 and N4
   2 – node N5 is between N4 and N1
   3 – node N5 is between N1 and N2
   4 – node N5 is between N2 and N3
NX x-direction looping parameter, set equal to unity by STAGS if omitted or nonpositive; NX > 1 instructs STAGS to generate a set of NX type E845 elements in the x direction, using the increments given on the T-845g record
NY y-direction looping parameter, set equal to unity by STAGS if omitted or nonpositive; NY > 1 instructs STAGS to generate a set of NY type E845 elements in the y direction for each of the NX elements generated in the x direction in the current definition, using the increments given on the T-845h record
USER user-specified element number for the first E845 element generated here (used only if IUWLE = 1 on H-1)
T-845a E845 Lower Face-Sheet Properties

A single T-845a record must be included immediately following the T-845 record, to specify the nodes for the lower face sheet of the first E845 element and to specify other parameters for the lower face sheet of each of the element(s) specified in the current definition.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
<th>N5</th>
<th>IFABL</th>
<th>ZETAL</th>
<th>ECZL</th>
<th>IPLASL</th>
<th>IANGL</th>
</tr>
</thead>
</table>
| N1  | first node point on lower face sheet
| N2  | second node point on lower face sheet
| N3  | third node point on lower face sheet
| N4  | fourth node point on lower face sheet
| N5  | fifth node point on lower face sheet
| IFABL | fabrication identifier for the lower face sheet element(s); this is equivalent to the WALL parameter for E410 quadrilateral elements:
| >0 | wall configuration number in the Wall Fabrication Table (K-1)
| 0  | shell wall properties are given in user-written subroutine WALL
   | (See the T-4 record description for more information about this)
| <0 | shell fabrication identifier in the GCP Fabrication Table (I-21a)
   | (this option is not operational yet and should not be used)
| ZETAL | angle \( \zeta \) between the wall-reference coordinate \( x_w \) and the fabrication coordinate \( \bar{z} \), for the lower face sheet; \( \zeta \) is a right-handed rotation about \( \bar{z} \); see Figure 8.2 on page 8-19
| ECZL | eccentricity in \( z' \) direction, for the lower face sheet; ECZL is the \( z' \) coordinate of the shell wall middle surface; please refer to Figure 6.2 on page 6-28; in element units, \( (x', y', z') \) are used in place of \( (X', Y', Z') \), which do not exist in an element unit; see “Effects of Eccentricity” on page 16-6
| IPLASL | lower face sheet material-nonlinearity flag:
| 0  | elastic behavior only
| 1  | plasticity included, with the material law satisfied at each element integration point
| 2  | plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)
IANGL lower face sheet wall-reference option; see discussion on page 8-18

0 – use default strategy of projecting $x_g, y_g$ to establish $x_w$

1 – input $r_w$, which is projected to establish $x_w$; see (T-845b)

if ( IANGL > 0 ) then
  go to T-845b
else
  go to T-845c
T-845b E845 Lower Face-Sheet Wall Reference Vector

RXL, RYL, RZL

lower face sheet wall reference vector, \( \mathbf{r}_w \), which is projected onto the element surface to determine the direction of the wall-reference coordinate \( x_w \); \( \mathbf{r}_w \) is expressed in \( (x_g, y_g, z_g) \) global coordinates

\[ \text{go to T-845c} \]
**T-845c E845 Upper Face-Sheet Properties**

A single T-845c record must be included immediately following the T-845a (or T-845b) record, to specify the nodes for the upper face sheet of the first E845 element and to specify other parameters for the upper face sheet of each of the element(s) specified in the current definition.

---

<table>
<thead>
<tr>
<th>N6</th>
<th>N7</th>
<th>N8</th>
<th>N9</th>
<th>N10</th>
<th>IFABU</th>
<th>ZETAU</th>
<th>ECZU</th>
<th>IPLASU</th>
<th>IANGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>first node point on upper face sheet</td>
<td>second node point on upper face sheet</td>
<td>third node point on upper face sheet</td>
<td>fourth node point on upper face sheet</td>
<td>fifth node point on upper face sheet</td>
<td>fabrication identifier for the upper face sheet element(s); this is equivalent to the WALL parameter for E410 quadrilateral elements:</td>
<td>angle $\zeta$ between the wall-reference coordinate $x_w$ and the fabrication coordinate $x$, for the upper face sheet; $\zeta$ is a right-handed rotation about $z$</td>
<td>eccentricity in $z'$ direction, for the upper face sheet; ECZL is the $z'$ coordinate of the shell wall middle surface</td>
<td>upper face sheet material-nonlinearity flag:</td>
<td>upper face sheet wall-reference option; see discussion on page 8-18</td>
</tr>
<tr>
<td>$&gt;0$ – wall configuration number in the Wall Fabrication Table (K-1)</td>
<td>$0$ – shell wall properties are given in user-written subroutine WALL</td>
<td>$&lt;0$ – shell fabrication identifier in the GCP Fabrication Table (I-21a)</td>
<td>0 – elastic behavior only</td>
<td>1 – plasticity included, with the material law satisfied at each element integration point</td>
<td>2 – plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)</td>
<td>0 – use default strategy of projecting $x_{g'}, y_g$ to establish $x_w$</td>
<td>1 – input $r_w$, which is projected to establish $x_w$; see (T-845d)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```plaintext
if ( IANGL > 0 ) then go to T-845d
else go to T-845e
```
T-845d E845 Upper Face-Sheet Wall Reference Vector

RXU, RYU, RZU

upper face sheet wall reference vector, $\mathbf{r}_w$, which is projected onto the element surface to determine the direction of the wall-reference coordinate $x_w$; $\mathbf{r}_w$ is expressed in $(x_g, y_g, z_g)$ global coordinates

go to T-845e
**T-845e E845 Core Properties**

A single T-845e record must be included immediately following the T-845c (or T-845d) record, to specify parameters for the core component of the element(s) specified in the current definition.

---

**IFABC ZETAC IPLASC IANGC**

- **IFABC**
  - core fabrication identifier:
    - \( > 0 \) – wall configuration number in the Wall Fabrication Table (K-1)
    - \( < 0 \) – solid fabrication identifier in the GCP Fabrication Table (I-22a)

- **ZETAC**
  - angle \( \zeta \) between the wall-reference coordinate \( x_w \) and the fabrication coordinate \( \bar{x} \), for the core component; \( \zeta \) is a right-handed rotation about \( \bar{z} \)

- **IPLASC**
  - core material-nonlinearity flag:
    - \( 0 \) – elastic behavior only
    - \( 1 \) – plasticity included (not operational with GCP fabrications, yet)

- **IANGC**
  - core wall-reference option; see discussion on page 8-18
    - \( 0 \) – use default strategy of projecting \( x_{e'}, y_{e'} \) to establish \( x_w \)
    - \( 1 \) – input \( r_w \), which is projected to establish \( x_w \); see (T-845f)

- **NX (T-845)**
  - x-direction looping parameter

- **NY (T-845)**
  - y-direction looping parameter

if ( **IANGC > 0** ) then *go to T-845f*
elseif ( **NX > 0** ) then *go to T-845g*
elseif ( **NY > 0** ) then *go to T-845h*
else *return to T-100*
T-845f E845 Core Reference Vector

**RXC  RYC  RZC**

RXC, RYC, RZC  
core reference vector, \( \mathbf{r}_w \), which is projected onto the element surface to determine the direction of the core-reference coordinate \( x_w \); \( \mathbf{r}_w \) is expressed in \( (x_g, y_g, z_g) \) global coordinates

**NX**  (T-845)  
x-direction looping parameter

**NY**  (T-845)  
y-direction looping parameter

if \( (\ NX > 0 \ ) \) then go to T-845g
elseif \( (\ NY > 0 \ ) \) then go to T-845h
else return to T-100
**T-845g E845 X-Direction Incrementations**

A single record of type T-845g must be included immediately after the T-845e or T-845f record when the **nx** “x-direction” looping parameter (T-845) is greater than unity. Ten nodal incrementation variables are specified here for use with the “x-direction” looping function on the T-845 record. nx E845 sandwich transition elements are generated in the x direction with this information. One incrementation variable is also specified here for “x-direction” looping with the user-element-number parameter on T-845.

<table>
<thead>
<tr>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>I4</th>
<th>I5</th>
<th>I6</th>
<th>I7</th>
<th>I8</th>
<th>I9</th>
<th>I10</th>
<th>I11</th>
</tr>
</thead>
</table>

I1 incrementation for the N1 lower face sheet node on the T-845a record  
I2 incrementation for the N2 lower face sheet node on the T-845a record  
I3 incrementation for the N3 lower face sheet node on the T-845a record  
I4 incrementation for the N4 lower face sheet node on the T-845a record  
I5 incrementation for the N5 lower face sheet node on the T-845a record  
I6 incrementation for the N6 upper face sheet node on the T-845c record  
I7 incrementation for the N7 upper face sheet node on the T-845c record  
I8 incrementation for the N8 upper face sheet node on the T-845c record  
I9 incrementation for the N9 upper face sheet node on the T-845c record  
I10 incrementation for the N10 upper face sheet node on the T-845c record  
I11 incrementation for USER on the T-845 record

Any of these incrementation variables can be negative, zero, or positive, as required.

**Example:** the following T-845 / T-845a / T-845c / T-845g record combination:

```
845 1 0 0 1 3 $ T-845 with NX=3
10 20 30 40 50 ... $ T-845a lower face nodes
60 70 80 90 100 ... $ T-845c upper face nodes
...  
1 1 1 1 1 2 2 2 2 $ T-845g increments
```

generates three E845 elements, with the following nodes:

```
10 20 30 40 50 60 70 80 90 100 Element #1
11 21 31 41 51 62 72 82 92 102 Element #2
12 22 32 42 52 64 74 84 94 104 Element #3
```

\[
NY \quad \text{(T-845)} \quad \text{y-direction looping parameter}
\]

if \quad \text{(NY > 0)} \quad \text{then} \quad \text{go to T-845h}
else \quad \text{return to T-100}
**T-845h Y-Direction Incrementations**

A type T-845h record must be included immediately after the T-845e or T-845f or T-845g record when the **NY** “y-direction” **looping parameter** is greater than unity (T-845). Ten nodal incrementation variables are specified here for use with the T-845 record “y-direction” looping function. **NY E845** sandwich transition elements are generated in the “y-direction”, for each **E845** element generated in the x direction, with this information. The **NX** and **NY** looping parameters are usually employed together to specify **NX × NY** **E845** elements in a single stroke.

<table>
<thead>
<tr>
<th>J1</th>
<th>J2</th>
<th>J3</th>
<th>J4</th>
<th>J5</th>
<th>J6</th>
<th>J7</th>
<th>J8</th>
<th>J9</th>
<th>J10</th>
<th>J11</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1</td>
<td>incrementation for the <strong>N1</strong> lower face sheet node on the T-845a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J2</td>
<td>incrementation for the <strong>N2</strong> lower face sheet node on the T-845a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J3</td>
<td>incrementation for the <strong>N3</strong> lower face sheet node on the T-845a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J4</td>
<td>incrementation for the <strong>N4</strong> lower face sheet node on the T-845a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J5</td>
<td>incrementation for the <strong>N5</strong> lower face sheet node on the T-845a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J6</td>
<td>incrementation for the <strong>N6</strong> upper face sheet node on the T-845c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J7</td>
<td>incrementation for the <strong>N7</strong> upper face sheet node on the T-845c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J8</td>
<td>incrementation for the <strong>N8</strong> upper face sheet node on the T-845c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J9</td>
<td>incrementation for the <strong>N9</strong> upper face sheet node on the T-845c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J10</td>
<td>incrementation for the <strong>N10</strong> upper face sheet node on the T-845c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J11</td>
<td>incrementation for <strong>USER</strong> on the T-845 record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero, or positive, as required.

**Example:** the following **T-845 / T-845a / T-845c / T-845g / T-845h** record combination:

```
845 1 0 0 1 3 2 $ T-845 NX=3, NY=2
10 20 30 40 50 ... $ T-845a (lower face nodes)
60 70 80 90 100 ... $ T-845c (upper face nodes)
...
1 1 1 1 1 1 1 1 1 1 $ T-845g (x increments)
5 5 5 5 5 5 5 5 5 5 $ T-845h (y increments)
```

generates six **E845** elements, with the following nodes:

```
10 20 30 40 50 60 70 80 90 100 Element #1
11 21 31 41 51 61 71 81 91 101 Element #2
12 22 32 42 52 62 72 82 92 102 Element #3
15 25 35 45 55 65 75 85 95 105 Element #4
16 26 36 46 56 66 76 86 96 106 Element #5
17 27 37 47 57 67 77 87 97 107 Element #6
```

*return to T-100*
**T-847 14-Node Sandwich Transition Element Definition**

The **E847** 14-node sandwich-transition element, shown in Figure 9.6, is used in sandwich shell units to make a transition between one mesh and two other meshes that have nodes that are exactly half as far apart as the one in which the **E847** element “resides.” It can be used to accomplish the same objective in an element unit. A single T-847 definition may specify one or more **E847** sandwich transition elements.

![E847 14-Node Sandwich Transition Elements](image)

**Figure 9.6  E847 14-Node Sandwich Transition Elements**

Definition of one or more **E847** 14-node sandwich transition elements is accomplished with the following set of records:

- **T-847** to specify parameters used for all parts of the element
- **T-847a** to specify information for the lower face sheet of the element
- **T-847b** to specify lower face sheet transformation angles (optional)
- **T-847c** to specify information for the upper face sheet
- **T-847d** to specify upper face sheet transformation angles (optional)
- **T-847e** to specify core parameters
- **T-847f** to specify core transformation angles (optional)
- **T-847g** for “x-direction” incrementation parameters
- **T-847h** for “y-direction” incrementation parameters

These are described sequentially in the following text.
KEL T ILIN INTEG IPEN IEDGE NX NY USER

ILIN  geometric-linearity flag:

0 – nonlinear strain-displacement relations
1 – linear strain-displacement relations

INTEG  integration-type flag (see N-1):

0 – standard integration, or 2 x 2 Gauss points for E410 elements
1 – modified 5-point integration, previously referred to as full integration

IPEN  penalty option (see N-1)

0 – no penalty function on fourth-order terms in E410 elements
1 – penalty function included in E410 elements

IEDGE  identifies the E847 edges on which nodes N5 and N6 are located

1 – node N5 is between N3 and N4; node N6 is between N4 and N1
2 – node N5 is between N4 and N1; node N6 is between N1 and N2
3 – node N5 is between N1 and N2; node N6 is between N2 and N3
4 – node N5 is between N2 and N3; node N6 is between N3 and N4

NX  x-direction looping parameter, set equal to unity by STAGS if omitted or nonpositive; NX > 1 instructs STAGS to generate a set of nx type E847 elements in the x direction, using the increments given on T-847g

NY  y-direction looping parameter, set equal to unity by STAGS if omitted or nonpositive; NY > 1 instructs STAGS to generate a set of ny type E847 elements in the y direction for each of the nx elements generated in the x direction in the current definition, using the increments given on T-847h

USER  user-specified element number for the first E847 element generated here (used only if IUWLE = 1 on H-1)

go to T-847a
T-847a E847 Lower Face-Sheet Properties

A single T-847a record must be included immediately following the T-847 record, to specify the nodes for the lower face sheet of the first E847 element and to specify other parameters for the lower face sheet of each of the element(s) specified in the current definition.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
<th>N5</th>
<th>N6</th>
<th>N7</th>
<th>IFABL</th>
<th>ZETAL</th>
<th>ECZL</th>
<th>IPLASL</th>
<th>IANGL</th>
</tr>
</thead>
</table>

N1  first node point on lower face sheet
N2  second node point on lower face sheet
N3  third node point on lower face sheet
N4  fourth node point on lower face sheet
N5  fifth node point on lower face sheet
N6  sixth node point on lower face sheet
N7  seventh node point on lower face sheet

IFABL  fabrication identifier for the lower face sheet element(s); this is equivalent to the WALL parameter for E410 quadrilateral elements:

>0  –  wall configuration number in the Wall Fabrication Table (K-1)
0  –  shell wall properties are given in user-written subroutine WALL
    (See the T-4 record description for more information about this)
<0  –  shell fabrication identifier in the GCP Fabrication Table (I-21a)

ZETAL  angle $\zeta$ between the wall-reference coordinate $x_w$ and the fabrication coordinate $x$, for the lower face sheet; $\zeta$ is a right-handed rotation about $z$; see Figure 8.2 on page 8-19.

ECZL  eccentricity in $\zeta'$ direction, for the lower face sheet; ECZL is the $\zeta'$ coordinate of the shell wall middle surface; please refer to Figure 6.2 on page 6-28; in element units, $(x', y', z')$ are used in place of $(X', Y', Z')$, which do not exist in an element unit; see “Effects of Eccentricity” on page 16-6.

IPLASL  lower face sheet material-nonlinearity flag:

0  –  elastic behavior only
1  –  plasticity included, with the material law satisfied at each element integration point
2  –  plasticity included, with the material law satisfied at the element centroid (centroïdal plasticity)
**IANGL**

lower face sheet wall-reference option; see discussion on page 8-18

0  – use default strategy of projecting $x_g, y_g$ to establish $x_w$

1  – input $r_w$, which is projected to establish $x_w$; see (T-847b)

```plaintext
if ( IANGL > 0 ) then
go to T-847b
else
go to T-847c
```
**T-847b E847 Lower Face-Sheet Wall Reference Vector**

**RXL RYL RZL**

lower face sheet wall reference vector, $\mathbf{r}_w$, which is projected onto the element surface to determine the direction of the wall-reference coordinate $x_w$; $\mathbf{r}_w$ is expressed in $(x_g, y_g, z_g)$ global coordinates.

go to **T-847c**
**T-847c E847 Upper Face-Sheet Properties**

A single T-847c record must be included immediately following the T-847a (or T-847b) record, to specify the nodes for the upper face sheet of the first E847 element and to specify other parameters for the upper face sheet of each of the element(s) specified in the current definition.

<table>
<thead>
<tr>
<th>N8</th>
<th>N9</th>
<th>N10</th>
<th>N11</th>
<th>N12</th>
<th>N13</th>
<th>N14</th>
<th>IFABU</th>
<th>ZETAU</th>
<th>ECZU</th>
<th>IPLASU</th>
<th>IANGU</th>
</tr>
</thead>
<tbody>
<tr>
<td>N8</td>
<td>first node point on upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N9</td>
<td>second node point on upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N10</td>
<td>third node point on upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N11</td>
<td>fourth node point on upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N12</td>
<td>fifth node point on upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N13</td>
<td>sixth node point on upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N14</td>
<td>seventh node point on upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFABU</td>
<td>fabrication identifier for the upper face sheet element(s); this is equivalent to the WALL parameter for E410 quadrilateral elements:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;0</td>
<td>wall configuration number in the Wall Fabrication Table (K-1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>shell wall properties are given in user-written subroutine WALL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;0</td>
<td>shell fabrication identifier in the GCP Fabrication Table (I-21a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZETAU</td>
<td>angle $\zeta$ between the wall-reference coordinate $x_w$ and the fabrication coordinate $x$, for the upper face sheet; $\zeta$ is a right-handed rotation about $\zeta$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECZU</td>
<td>eccentricity in $z'$ direction, for the upper face sheet; ECZL is the $z'$ coordinate of the shell wall middle surface</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPLASU</td>
<td>upper face sheet material-nonlinearity flag:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>elastic behavior only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>plasticity included, with the material law satisfied at each element integration point</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**IANGL**

upper face sheet wall-reference option; see discussion on page 8-18

0 – use default strategy of projecting $x_g, y_g$ to establish $x_w$

1 – input $r_w$, which is projected to establish $x_w$; see (T-847d)

```plaintext
if ( IANGU > 0 ) then  go to T-847d
else                    go to T-847e
```
**T-847d E847 Upper Face-Sheet Wall Reference Vector**

**RXU, RYU, RZU**

Upper face sheet wall reference vector, \( \mathbf{r}_w \), which is projected onto the element surface to determine the direction of the wall-reference coordinate \( x_w \); \( \mathbf{r}_w \) is expressed in \((x_g, y_g, z_g)\) global coordinates.

*go to T-847e*
## T-847e E847 Core Properties

A single T-847e record must be included immediately following the T-847c (or T-847d) record, to specify parameters for the core component of the element(s) specified in the current definition.

<table>
<thead>
<tr>
<th>IFABC</th>
<th>core fabrication identifier:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;0</td>
<td>wall configuration number in the Wall Fabrication Table (K-1)</td>
</tr>
<tr>
<td>&lt;0</td>
<td>solid fabrication identifier in the GCP Fabrication Table (I-22a)</td>
</tr>
</tbody>
</table>

| ZETAC | angle \( \zeta \) between the wall-reference coordinate \( x_w \) and the fabrication coordinate \( x \), for the core component; \( \zeta \) is a right-handed rotation about \( z \) |

<table>
<thead>
<tr>
<th>IPLASC</th>
<th>core material-nonlinearity flag:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>elastic behavior only</td>
</tr>
<tr>
<td>1</td>
<td>plasticity included (not operational with GCP fabrications, yet)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IANGC</th>
<th>core wall-reference option; see discussion on page 8-18</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>use default strategy of projecting ( x_w, y_g ) to establish ( x_w )</td>
</tr>
<tr>
<td>1</td>
<td>input ( r_w ), which is projected to establish ( x_w ); see (T-847f)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \text{NX} ) (T-847)</th>
<th>x-direction looping parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{NY} ) (T-847)</td>
<td>y-direction looping parameter</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{if} & \quad (\text{IANGC} > 0) \quad \text{then go to T-847f} \\
\text{elseif} & \quad (\text{NX} > 0) \quad \text{then go to T-847g} \\
\text{elseif} & \quad (\text{NY} > 0) \quad \text{then go to T-847h} \\
\text{else} & \quad \text{return to T-100}
\end{align*}
\]
**T-847f E847 Core Reference Vector**

**RXC, RYC, RZC**

core reference vector, \( \mathbf{r_w} \), which is projected onto the element surface to determine the direction of the core-reference coordinate \( x_w \); \( \mathbf{r_w} \) is expressed in \((x_g, y_g, z_g)\) global coordinates

\[
\begin{align*}
\text{if } (\text{NX} > 0) & \text{ then go to T-847g} \\
\text{elseif } (\text{NY} > 0) & \text{ then go to T-847h} \\
\text{else } & \text{return to T-100}
\end{align*}
\]
**T-847g X-Direction Incrementations**

A single record of type T-847g must be included immediately after the T-847e or T-847f record when the \textbf{nx} “x-direction” \textit{looping parameter} (T-847) is greater than unity. Fourteen nodal incrementation variables are specified here for use with the “x-direction” looping function on the T-847 record. \textbf{Nx} E847 sandwich elements are generated in the \textit{x} direction with this information. One incrementation variable is also specified here for “x-direction” looping with the user-element-number parameter on T-847.

\begin{center}
\begin{tabular}{cccccccccccccc}
\hline
\textbf{I1} & \textbf{I2} & \textbf{I3} & \textbf{I4} & \textbf{I5} & \textbf{I6} & \textbf{I7} & \textbf{I8} & \textbf{I9} & \textbf{I10} & \textbf{I11} & \textbf{I12} & \textbf{I13} & \textbf{I14} & \textbf{I15} \\
\hline
\end{tabular}
\end{center}

- I1 incrementation for the \textbf{N1} lower face sheet node on the T-847a record
- I2 incrementation for the \textbf{N2} lower face sheet node on the T-847a record
- I3 incrementation for the \textbf{N3} lower face sheet node on the T-847a record
- I4 incrementation for the \textbf{N4} lower face sheet node on the T-847a record
- I5 incrementation for the \textbf{N5} lower face sheet node on the T-847a record
- I6 incrementation for the \textbf{N6} lower face sheet node on the T-847a record
- I7 incrementation for the \textbf{N7} lower face sheet node on the T-847a record
- I8 incrementation for the \textbf{N8} upper face sheet node on the T-847c record
- I9 incrementation for the \textbf{N9} upper face sheet node on the T-847c record
- I10 incrementation for the \textbf{N10} upper face sheet node on the T-847c record
- I11 incrementation for the \textbf{N11} upper face sheet node on the T-847c record
- I12 incrementation for the \textbf{N12} upper face sheet node on the T-847c record
- I13 incrementation for the \textbf{N13} upper face sheet node on the T-847c record
- I14 incrementation for the \textbf{N14} upper face sheet node on the T-847c record
- I15 incrementation for \textbf{USER} on the T-847 record

Any of these incrementation variables can be negative, zero, or positive, as required.

\textbf{NY} (T-847) y-direction looping parameter

\begin{align*}
\text{if} & \quad (\textbf{NY} > 0) \quad \text{then} \quad \text{go to T-847h} \\
\text{else} & \quad \text{return to T-100}
\end{align*}
T-847h E847 Y-Direction Incremenations

A type T-847h record must be included immediately after the T-847e or T-847f or T-847g record when the NY “y-direction” looping parameter is greater than unity (T-847). Fourteen nodal incrementation variables are specified here for use with the T-847 record “y-direction” looping function. NY E847 sandwich elements are generated in the y direction, for each E847 element generated in the x direction, with this information. The NX and NY looping parameters are usually employed together to specify NX × NY E847 elements in a single stroke.

<table>
<thead>
<tr>
<th>J1</th>
<th>J2</th>
<th>J3</th>
<th>J4</th>
<th>J5</th>
<th>J6</th>
<th>J7</th>
<th>J8</th>
<th>J9</th>
<th>J10</th>
<th>J11</th>
<th>J12</th>
<th>J13</th>
<th>J14</th>
<th>J15</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1</td>
<td>incrementation for the N1 lower face sheet node on the T-847a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J2</td>
<td>incrementation for the N2 lower face sheet node on the T-847a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J3</td>
<td>incrementation for the N3 lower face sheet node on the T-847a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J4</td>
<td>incrementation for the N4 lower face sheet node on the T-847a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J5</td>
<td>incrementation for the N5 lower face sheet node on the T-847a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J6</td>
<td>incrementation for the N6 lower face sheet node on the T-847a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J7</td>
<td>incrementation for the N7 lower face sheet node on the T-847a record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J8</td>
<td>incrementation for the N8 upper face sheet node on the T-847c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J9</td>
<td>incrementation for the N9 upper face sheet node on the T-847c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J10</td>
<td>incrementation for the N10 upper face sheet node on the T-847c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J11</td>
<td>incrementation for the N11 upper face sheet node on the T-847c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J12</td>
<td>incrementation for the N12 upper face sheet node on the T-847c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J13</td>
<td>incrementation for the N13 upper face sheet node on the T-847c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J14</td>
<td>incrementation for the N14 upper face sheet node on the T-847c record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J15</td>
<td>incrementation for USER on the T-847 record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero, or positive, as required.
**T-849 E849 18-Node Sandwich Element Definition**

The **E849** 18-node sandwich element in **STAGS**, shown in Figure 9.7, is constructed with a pair of **E480** 9–node quadrilateral shells (with bending as well as the usual membrane stiffness) that are held apart by a lightweight core that has three-dimensional elastic properties.

![E849 18-Node Sandwich Element](image)

**Figure 9.7** **E849** 18-Node Sandwich Element

Definition of one or more **E849** 18-node sandwich elements is accomplished with the following set of records:

- **T-849** to specify parameters used for all parts of the element
- **T-849a** to specify information for the lower face sheet of the element
- **T-849b** to specify lower face sheet transformation angles (optional)
- **T-849c** to specify information for the upper face sheet
- **T-849d** to specify upper face sheet transformation angles (optional)
- **T-849e** to specify core parameters
- **T-849f** to specify core transformation angles (optional)
- **T-849g** for “x-direction” incrementation parameters
- **T-849h** for “y-direction” incrementation parameters

These are described sequentially in the following text.
**Definition of Sandwich Elements via the Ecom Protocol Model Input—Element Units (3)**

<table>
<thead>
<tr>
<th>KELT</th>
<th>element code number (must be 849)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILIN</td>
<td>geometric-linearity flag:</td>
</tr>
<tr>
<td></td>
<td>0 – nonlinear strain-displacement relations</td>
</tr>
<tr>
<td></td>
<td>1 – linear strain-displacement relations</td>
</tr>
<tr>
<td>INTEG</td>
<td>integration-type flag: not used for this element; set INTEG = 0</td>
</tr>
<tr>
<td>IPEN</td>
<td>penalty option: not used for this element; set IPEN = 0</td>
</tr>
<tr>
<td>NX</td>
<td>x-direction looping parameter, set equal to unity by STAGS if omitted or nonpositive; ( \text{NX} &gt; 1 ) instructs STAGS to generate a set of ( \text{NX} ) type ( \text{E849} ) elements in the ( x ) direction, using nodal incrementation information given on ( \text{T-849g} )</td>
</tr>
<tr>
<td>NY</td>
<td>y-direction looping parameter, set equal to unity by STAGS if omitted or nonpositive; ( \text{NY} &gt; 1 ) instructs STAGS to generate a set of ( \text{NY} ) type ( \text{E849} ) elements in the ( y ) direction for each of the ( \text{NX} ) elements generated in the ( x ) direction in the current definition, using nodal incrementation information given on ( \text{T-849h} )</td>
</tr>
<tr>
<td>USERC</td>
<td>user-specified element number for core component</td>
</tr>
<tr>
<td></td>
<td>(used only if ( \text{IUWLE} = 1 ) on H-1)</td>
</tr>
<tr>
<td>USER1</td>
<td>user-specified element number for lower face sheet, (if ( \text{IUWLE} = 1 ) on H-1)</td>
</tr>
<tr>
<td>USER2</td>
<td>user-specified element number for upper face sheet, (if ( \text{IUWLE} = 1 ) on H-1)</td>
</tr>
</tbody>
</table>

* go to \( \text{T-849a} \)
T-849a E849 Lower Face-Sheet Properties

A single T-849a record must be included immediately following the T-849 record, to specify the nodes for the lower face sheet of the first E849 element (see Figure 9.7 on page 9-120) and to specify other parameters for the lower face sheet of each element that is specified in the current definition.

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
<th>N5</th>
<th>N6</th>
<th>N7</th>
<th>N8</th>
<th>N9</th>
<th>IFABL</th>
<th>ZETAL</th>
<th>ECZL</th>
<th>IPLASL</th>
<th>IANGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>node # 1 on the lower face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>node # 2 on the lower face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>node # 3 on the lower face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N4</td>
<td>node # 4 on the lower face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N5</td>
<td>node # 5 on the lower face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N6</td>
<td>node # 6 on the lower face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N7</td>
<td>node # 7 on the lower face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N8</td>
<td>node # 8 on the lower face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N9</td>
<td>node # 9 on the lower face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IFABL fabrication identifier for the lower face sheet element(s); this is equivalent to the WALL parameter for E480 quadrilateral elements:

- >0 – wall configuration number in the Wall Fabrication Table (K-1)
- 0 – shell wall properties are given in user-written subroutine WALL (See the T-4 record description for more information about this)
- <0 – shell fabrication identifier in the GCP Fabrication Table (I-21a)

ZETAL angle ζ between the wall-reference coordinate x_w and the fabrication coordinate x̃, for the lower face sheet; ζ is a right-handed rotation about z̃; see Figure 8.2 on page 8-19.

ECZL eccentricity in z’ direction, for the lower face sheet; ECZL is the z’ coordinate of the shell wall middle surface; please refer to Figure 6.2 on page 6-28; in element units, (x’, y’, z’) are used in place of (X’, Y’, Z’), which do not exist in an element unit; see “Effects of Eccentricity” on page 16-6.

IPLASL lower face sheet material-nonlinearity flag:

- 0 – elastic behavior only
- 1 – plasticity included, with the material law satisfied at each element integration point
- 2 – plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)
IANGL: lower face sheet wall-reference option; see discussion on page 8-18

0 – use default strategy of projecting \( x_g', y_g' \) to establish \( x_w \)
1 – input \( r_w \), which is projected to establish \( x_w \); see (T-849b)

\[
\text{if} \ ( \text{IANGL} > 0 ) \ \text{then} \quad \text{go to T-849b}
\]
\[
\text{else} \quad \text{go to T-849c}
\]
**T-849b E849 Lower Face-Sheet Wall Reference Vector**

**RXL RYL RZL**

RXL, RYL, RZL lower face sheet wall reference vector, \( \mathbf{r}_w \), which is projected onto the element surface to determine the direction of the wall-reference coordinate \( x_w \); \( \mathbf{r}_w \) is expressed in \((x_g, y_g, z_g)\) global coordinates.

*go to T-849c*
**T-849c E849 Upper Face-Sheet Properties**

A single T-849c record must be included immediately following the T-849a (or T-849b) record, to specify the nodes for the upper face sheet of the first E849 element (see Figure 9.7 on page 9-120) and to specify other parameters for the upper face sheet of each element that is specified in the current definition.

<table>
<thead>
<tr>
<th>N10</th>
<th>N11</th>
<th>N12</th>
<th>N13</th>
<th>N14</th>
<th>N15</th>
<th>N16</th>
<th>N17</th>
<th>N18</th>
<th>IFABU</th>
<th>ZETAU</th>
<th>ECZU</th>
<th>IPLASU</th>
<th>IANGU</th>
</tr>
</thead>
<tbody>
<tr>
<td>N10</td>
<td>node # 1 on the upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N11</td>
<td>node # 2 on the upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N12</td>
<td>node # 3 on the upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N13</td>
<td>node # 4 on the upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N14</td>
<td>node # 5 on the upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N15</td>
<td>node # 6 on the upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N16</td>
<td>node # 7 on the upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N17</td>
<td>node # 8 on the upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N18</td>
<td>node # 9 on the upper face sheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFABU</td>
<td>fabrication identifier for the upper face sheet element(s); this is equivalent to the WALL parameter for E410 quadrilateral elements:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;0</td>
<td>wall configuration number in the Wall Fabrication Table (K-1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>shell wall properties are given in user-written subroutine WALL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;0</td>
<td>shell fabrication identifier in the GCP Fabrication Table (I-21a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZETAU</td>
<td>angle ( \zeta ) between the wall-reference coordinate ( x_w ) and the fabrication coordinate ( x ), for the upper face sheet; ( \zeta ) is a right-handed rotation about ( \zeta )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECZU</td>
<td>eccentricity in ( z' ) direction, for the upper face sheet; ECZL is the ( z' ) coordinate of the shell wall middle surface</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPLASU</td>
<td>upper face sheet material-nonlinearity flag:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>elastic behavior only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>plasticity included, with the material law satisfied at each element integration point</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IANGU</td>
<td>upper face sheet wall-reference option; see discussion on page 8-18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>use default strategy of projecting ( x_{wp}, y_{wp} ) to establish ( x_w )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>input ( r_w ), which is projected to establish ( x_w ); see (T-849d)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{if ( IANGU > 0 ) then } \quad \text{go to T-849d} \\
\text{else} \quad \text{go to T-849e}
\]
T-849d E849 Upper Face-Sheet Wall Reference Vector

RXU, RYU, RZU

upper face sheet wall reference vector, \( \mathbf{r}_w \), which is projected onto the element surface to determine the direction of the wall-reference coordinate \( x_w \); \( \mathbf{r}_w \) is expressed in \((x_g, y_g, z_g)\) global coordinates

---

*go to* T-849e
**T-849e E849 Core Properties**

A single T-849e record must be included immediately following the T-849c (or T-849d) record, to specify parameters for the core component of the element(s) specified in the current definition.

---

**IFABC ZETAC IPLASC IANGC**

- **IFABC**
  - core fabrication identifier:
    - >0 – wall configuration number in the Wall Fabrication Table (K-1)
    - <0 – solid fabrication identifier in the GCP Fabrication Table (I-22a)

- **ZETAC**
  - angle $\zeta$ between the wall-reference coordinate $x_w$ and the fabrication coordinate $x$, for the core component; $\zeta$ is a right-handed rotation about $\zeta$

- **IPLASC**
  - core material-nonlinearity flag:
    - 0 – elastic behavior only
    - 1 – plasticity included (*not operational with GCP fabrications, yet*)

- **IANGC**
  - core wall-reference option; see discussion on page 8-18
    - 0 – use default strategy of projecting $x'_w, y'_g$ to establish $x_w$
    - 1 – input $r_w$, which is projected to establish $x_w$; see (T-849f)

- **NX** (T-849) x-direction looping parameter
- **NY** (T-849) y-direction looping parameter

```plaintext
if ( IANGC > 0 ) then go to T-849f
elseif ( NX > 0 ) then go to T-849g
elseif ( NY > 0 ) then go to T-849h
else return to T-100
```
**T-849f E849 Core Reference Vector**

**RXC, RYC, RZC**

The core reference vector, \( \mathbf{r}_w \), which is projected onto the element surface to determine the direction of the core-reference coordinate \( x_w \); \( \mathbf{r}_w \) is expressed in \( (x_g, y_g, z_g) \) global coordinates.

\[
\begin{align*}
\text{NX} & \quad (\text{T-849}) \quad \text{x-direction looping parameter} \\
\text{NY} & \quad (\text{T-849}) \quad \text{y-direction looping parameter}
\end{align*}
\]

if \( \text{NX} > 0 \) then \text{go to T-849g}
elseif \( \text{NY} > 0 \) then \text{go to T-849h}
else \text{return to T-100}
**T-849g X-Direction Incrementations**

A single record of type T-849g must be included immediately after the T-849e or T-849f record when the NX “x-direction” looping parameter (T-849) is greater than unity. Eighteen nodal incrementation variables are specified here for use with the “x-direction” looping function on the T-849 record. **NX E849** sandwich elements are generated in the x direction with this information. Three incrementation variables are also specified here for “x-direction” looping with the user-element-number parameters on T-849.

\[
( \text{IX(k), } k=1,18 ) \quad \text{IXC} \quad \text{IX1} \quad \text{IX2}
\]

- **IX(k)**: x-direction incrementation for the kth node, which is on the T-849a record (when \(1 \leq k \leq 9\)), or on the T-849c record (when \(10 \leq k \leq 18\))
- **IXC**: x-direction incrementation for **USERC** on the T-849 record
- **IX1**: x-direction incrementation for **USER1** on the T-849 record
- **IX2**: x-direction incrementation for **USER2** on the T-849 record

Any of these incrementation variables can be negative, zero, or positive, as required.

**Example:** the following T-849 / T-849a / T-849c / T-849g record combination:

```
849 1 0 0 3 $ T-849 with NX=3
10 20 30 40 50 60 70 80 90 ... $ T-849a lower face nodes
15 25 35 45 55 65 75 85 95 ... $ T-849c upper face nodes
...
1 1 1 1 1 1 1 1 1 9*2 $ T-849g increments
```

generates three E849 elements, with the following nodes:

```
10 20 30 40 50 60 70 80 90 15 25 35 45 55 65 75 85 95 Element #1
11 21 31 41 51 61 71 81 91 17 27 37 47 57 67 77 87 97 Element #2
12 22 32 42 52 62 72 82 92 19 29 39 49 59 69 79 89 99 Element #3
```

**NY** (T-849) y-direction looping parameter

if \((\text{NY} > 0)\) then **go to T-849h**
else **return to T-100**
**T-849h Y-Direction Incrementations**

A single type T-849h record must be included immediately after the T-849e or T-849f or T-849g record when the **NY** “y-direction” looping parameter is greater than unity (T-849). Eighteen nodal incrementation variables are specified here for use with the T-849 record “y-direction” looping function. **NY E849** sandwich elements are generated in the y direction, for each E849 element generated in the x direction, with this information. The **NX** and **NY** looping parameters are usually employed together to specify **NX × NY E849** elements in a single stroke.

\[
\begin{align*}
( IY(k), \ k=1,18 ) & \quad IY C \ IY 1 \ IY 2 \\
IY(k) & \quad \text{y-direction incrementation for the kth node, which is} \\
& \quad \text{on the T-849a record (when } 1 \leq k \leq 9 \text{), or} \\
& \quad \text{on the T-849c record (when } 10 \leq k \leq 18 \text{) } \\
IY C & \quad \text{y-direction incrementation for USERC on the T-849 record} \\
IY 1 & \quad \text{y-direction incrementation for USER1 on the T-849 record} \\
IY 2 & \quad \text{y-direction incrementation for USER2 on the T-849 record} \\
\end{align*}
\]

Any of these incrementation variables can be negative, zero, or positive, as required.

* return to T-100
9.7 **Definition of Solid Elements via the Ecom Protocol**

The current version of **STAGS** has four conventional solid elements that are fully operational and “hooks” for an as-yet-undetermined number of unconventional solid elements that are not “ready for prime time” yet. The four operational solid elements—members of the **E880** “family” of elements—are shown in Figure 9.8. The other solid elements—members of the **E860** family—were originally implemented in and are being transferred into **STAGS** from the tetrahedral, pentahedral and hexahedral constructions in the ORACLE program.

![Figure 9.8 The E880 Family of Solid Elements](image)

In the Edef protocol (which was described in Chapter 8), the number of **E880**-element-family *definitions* to be made for the current element unit was specified *via* the **N880** parameter on the T-5 record and that the element *type* for any given definition was specified by the value of **KELT** in that definition. In the Ecom protocol (which is being described here), the command keyword (on T-100) specifies the element type explicitly.
The 8-node **E881** ANS solid element has 4 nodes on its lower face and 4 nodes on its upper face. One or more **E881** elements (or additional **E881** elements) can be defined with the T-881 record set. For more information about the **E881** element, see “E881 8-Node ANS solid element” on page 14-47.

The 18-node **E882** ANS solid element has 9 nodes on its lower face and 9 nodes on its upper face. One or more **E882** elements (or additional **E882** elements) can be defined with the T-882 record set. For more information about the **E882** element, see “E882 18-Node solid element” on page 14-48.

The 27-node **E883** ANS solid element, has 9 nodes on its lower face, 9 nodes on its mid-surface, and 9 nodes on its upper face. One or more **E883** elements (or additional **E883** elements) can be defined with the T-883 record set. For more information about the **E883** element, see “E883 27-Node solid element” on page 14-49.

The 20-node **E885** displacement-based solid element has 8 nodes on its lower face, 4 nodes on its mid-surface, and 8 nodes on its upper face. One or more **E885** elements (or additional **E885** elements) can be defined with the T-885 record set. For more information about the **E885** element, see “E885 20-Node displacement-based solid element” on page 14-50.
The E860 (ORACLE solid) element in STAGS, shown in Figure 9.9, is based on the general tetrahedral, pentahedral, or hexahedral element that was implemented in the ORACLE program.

The T-860 record is the first record of a multi-record set that defines one or more E860 solid elements to be included in the current element unit. The looping capabilities provided here, via NX and NY on T-860 and the incrementation parameters on T-860b and T-860c, can be used effectively in many situations.
<table>
<thead>
<tr>
<th>KELT</th>
<th>number of element sets to be generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNODES</td>
<td>number of node points required to define the current E860 element(s); NNODES must be greater than or equal to 4</td>
</tr>
<tr>
<td>IFAB</td>
<td>fabrication identifier for the element: &lt;0 – fabrication identifier in the GCP Fabrication Table (l-22a)</td>
</tr>
<tr>
<td>IANG</td>
<td>wall-reference option: 0 – use default strategy of projecting ( x_e, y_e ) to establish ( x_w ) 1 – input ( r_w ), which is projected to establish ( x_w )</td>
</tr>
<tr>
<td>ILIN</td>
<td>geometric-nonlinearity flag: 0 – use nonlinear strain-displacement relations 1 – use linear strain-displacement relations</td>
</tr>
<tr>
<td>IPLAS</td>
<td>material-nonlinearity flag: 0 – elastic behavior only 1 – plasticity included, with the material law satisfied at each element integration point 2 – plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)</td>
</tr>
<tr>
<td>NX</td>
<td>number of elements to be generated in the ( x ) direction; STAGS sets ( NX = 1 ) if it is nonpositive</td>
</tr>
<tr>
<td>NY</td>
<td>number of elements to be generated in the ( y ) direction; STAGS sets ( NY = 1 ) if it is nonpositive</td>
</tr>
</tbody>
</table>

*go to T-860b*
**T-860a E860 Solid Element Nodes**

A single T-860a record must follow the T-860 record, to identify the \texttt{NNODES} (T-860) nodes (S-1) that define the first ORACLE solid element of the set to be generated here.

\[
\text{(NODE(k), k=1,NNODES)} \quad \text{USERELT}
\]

- **NODE(k)**  \( k^{th} \) of \texttt{NNODES} node points for the initial \texttt{E860} solid element
- **USERELT** user-specified element number, used only if \texttt{I华润LE} = 1 on \texttt{H-1}

- **NX** (T-860)  x-direction incrementation flag
- **NY** (T-860)  y-direction incrementation flag
- **IANG** (T-860)  wall-reference option

\[
\begin{align*}
\text{if} & \quad (\text{NX} > 1) \quad \text{then} \quad \text{go to T-860b} \\
\text{elseif} & \quad (\text{NY} > 1) \quad \text{then} \quad \text{go to T-860c} \\
\text{elseif} & \quad (\text{IANG} > 0) \quad \text{then} \quad \text{go to T-860d} \\
\text{else} & \quad \text{return to T-100}
\end{align*}
\]
**T-860b E860 X-Direction Incrementations**

$\text{NNODES}+1$ (T-860) $x$-direction incrementation variables are specified here for use with the $\text{NX}$ looping function invoked on the T-860 record and the initial node points established on T-860a.

$$ (\text{IX}(k), k=1,\text{NNODES}) \quad \text{IXU} $$

- $\text{IX}(k)$: $x$-direction incrementation variable for node $\text{NODE}(k)$ on T-883a
- $\text{IXU}$: $x$-direction incrementation variable for use with $\text{USERELT}$

Any of these incrementation variables can be negative, zero or positive.

```plaintext
if ( NY > 1 ) then go to T-860c
elseif ( IANG = 1 ) then go to T-860d
else return to T-100
```
**T-860c E860 Y-Direction Incrementations**

**NNODES**+1 (T-860) y-direction incrementation variables are specified here for use with the **NY** looping function invoked on the T-860 record and the initial node points established on T-860a.

\[
(IY(k), \text{ } k=1, \text{NNODES} \) \text{ IYU}
\]

- **IY(k)**: y-direction incrementation variable for node **NODE(k)** on T-883a
- **IYU**: y-direction incrementation variable for use with **USERELT**

Any of these incrementation variables can be negative, zero or positive.

\[
\text{if } ( \text{ IANG } = 1 ) \text{ then go to T-860d}
\]

\[
\text{else return to T-100}
\]
**T-860d E860 Material Orientation Record**

If $\text{IANG} > 0$ (T-860), a single T-860d record must follow the T-860a record (or T-860b or T-860c record, if applicable) for the current E860 solid element. T-860d specifies the material orientation for each of the E860 elements that are to be generated by the current T-860 element-definition set.

\[
\begin{align*}
&XFX 
&XFY 
&XFZ 
&YFX 
&YFY 
&YFZ
\end{align*}
\]

- $XFX, XFY, XFZ$ vector components establishing the $x$ orientation of the material
- $YFX, YFY, YFZ$ vector components establishing the $y$ orientation of the material

*return to T-100*
**T-881 E881 8-Node Solid Element**

The E881 8-node ANS solid element, in STAGS, is shown in Figure 9.10:

![Figure 9.10 E881 8-Node ANS Solid Element](image)

The nodal ordering for E881 corresponds exactly to the ordering used in the PATRAN program.

The T-881 record is the first record of a multi-record set that defines one or more E881 8-node solid elements to be included in the current element unit. The looping capabilities provided here, via NX and NY on T-881 and the incrementation parameters on T-881b and T-881c, can be used effectively in many situations.
MODEL INPUT—ELEMENT UNITS (3)

Definition of Solid Elements via the Ecom Protocol

**KELT**  
fabrication identifier for the element; this is equivalent to the **WALL** parameter for **E410** quadrilateral elements:
- >0  – wall configuration number in the **Wall Fabrication Table** (K-1)
- 0  – wall properties are given in user-written subroutine **WALL**
- <0  – fabrication identifier in the **GCP Fabrication Table** (I-22a)

**IANG**  
wall-reference option:
- 0  – use default strategy of projecting \( x_g, y_g \) to establish \( x_w \)
- 1  – input \( r_w \), which is projected to establish \( x_w \)

**ILIN**  
geometric-nonlinearity flag:
- 0  – use nonlinear strain-displacement relations
- 1  – use linear strain-displacement relations

**IPLAS**  
material-nonlinearity flag:
- 0  – elastic behavior only
- 1  – plasticity included, with the material law satisfied at each element integration point
- 2  – plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)

**NX**  
number of elements to be generated in the \( x \) direction;
**STAGS** sets **NX** = 1 if it is nonpositive

**NY**  
number of elements to be generated in the \( y \) direction;
**STAGS** sets **NY** = 1 if it is nonpositive

> go to **T-881b**
**T-881a E881 Solid Element Nodes**

A single T-881a record must follow the T-881 record, to identify the 8 user nodes (S-1) that define the first solid element of the set to be generated here.

\[
( \text{NODE}(k), \ k=1,8 ) \quad \text{USERELT}
\]

- **NODE(k)**: \( k^{th} \) of 8 node points for the initial E881 solid element
- **USERELT**: user-specified element number, used only if \( \text{IUWLE} = 1 \) on H-1

- **NX** (T-881): x-direction incrementation flag
- **NY** (T-881): y-direction incrementation flag
- **IANG** (T-881): wall-reference option

```
if ( \text{NX} > 1 ) \text{ then go to T-881b }
elseif ( \text{NY} > 1 ) \text{ then go to T-881c }
elseif ( \text{IANG} > 0 ) \text{ then go to T-881d }
else \text{ return to T-100 }
```
T-881b E881 X-Direction Incrementations

Nine x-direction incrementation variables are specified here for use with the \texttt{NX} looping function invoked on the T-881 record and the initial node points established on T-881a.

\begin{tabular}{ll}
\textbf{IX1} & x-direction incrementation variable for node \texttt{N1} on T-881a \\
\textbf{IX2} & x-direction incrementation variable for node \texttt{N2} on T-881a \\
\textbf{IX3} & x-direction incrementation variable for node \texttt{N3} on T-881a \\
\textbf{IX4} & x-direction incrementation variable for node \texttt{N4} on T-881a \\
\textbf{IX5} & x-direction incrementation variable for node \texttt{N5} on T-881a \\
\textbf{IX6} & x-direction incrementation variable for node \texttt{N6} on T-881a \\
\textbf{IX7} & x-direction incrementation variable for node \texttt{N7} on T-881a \\
\textbf{IX8} & x-direction incrementation variable for node \texttt{N8} on T-881a \\
\textbf{IUX} & x-direction incrementation variable for use with \texttt{USERELT} \\
\end{tabular}

Any of these incrementation variables can be negative, zero or positive.

\begin{verbatim}
if ( \texttt{NY} > 1 ) then \texttt{go to T-881c}
elseif ( \texttt{IANG} = 1 ) then \texttt{go to T-881d}
else \texttt{return to T-100}
\end{verbatim}
**T-881c E881 Y-Direction Incrementations**

Nine y-direction incrementation variables are specified here for use with the *NY* looping function invoked on the T-881 record and the initial node points established on T-881a.

<table>
<thead>
<tr>
<th>IY1</th>
<th>IY2</th>
<th>IY3</th>
<th>IY4</th>
<th>IY5</th>
<th>IY6</th>
<th>IY7</th>
<th>IY8</th>
<th>IUY</th>
</tr>
</thead>
<tbody>
<tr>
<td>y-direction incrementation variable for node N1 on T-881a</td>
<td>y-direction incrementation variable for node N2 on T-881a</td>
<td>y-direction incrementation variable for node N3 on T-881a</td>
<td>y-direction incrementation variable for node N4 on T-881a</td>
<td>y-direction incrementation variable for node N5 on T-881a</td>
<td>y-direction incrementation variable for node N6 on T-881a</td>
<td>y-direction incrementation variable for node N7 on T-881a</td>
<td>y-direction incrementation variable for node N8 on T-881a</td>
<td>y-direction incrementation variable for use with USERELT</td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero or positive.

```plaintext
if ( IANG = 1 ) then go to T-881d  
else return to T-100
```
**T-881d E881 Material Orientation Record**

If $IANG > 0$ (T-881), a single T-881d record must follow the T-881a record (or T-881b or T-881c record, if applicable) for the current E881 solid element. T-881d specifies the material orientation for each of the E881 elements that are to be generated by the current T-881 element-definition set.

\[ \text{XFX XFY XFZ YFX YFY YFZ} \]

- **XFX, XFY, XFZ** vector components establishing the x orientation of the material
- **YFX, YFY, YFZ** vector components establishing the y orientation of the material

*return to T-100*
**T-882 E882 18-Node Solid Element**

The **E882** 18-node ANS solid element, in **STAGS**, is shown in Figure 9.11:

![Figure 9.11 E882 18-Node ANS Solid Element](image)

The nodal ordering for **E882** corresponds exactly to the ordering used in the **PATRAN** program.

The T-882 record is the first record of a multi-record set that defines one or more **E882** 18-node solid elements to be included in the current element unit. The *looping* capabilities provided here, *via* **NX** and **NY** on T-882 and the incrementation parameters on T-882b and T-882c, can be used effectively in many situations.
Model Input—Element Units (3)  

**Definition of Solid Elements via the Ecom Protocol**

---

### KELT  IFAB  IANG  ILIN  IPLAS  NX  NY

KELT  
**must = 882**

IFAB  
fabrication identifier for the element:

- **>0** – wall configuration number in the *Wall Fabrication Table* (K-1)
- **0** – wall properties are given in user-written subroutine WALL
- **<0** – fabrication identifier in the *GCP Fabrication Table* (I-22a)

IANG  
wall-reference option:

- **0** – use default strategy of projecting $x^g, y^g$ to establish $x_w$
- **1** – input $r_w$, which is projected to establish $x_w$

ILIN  
geometric-nonlinearity flag:

- **0** – use nonlinear strain-displacement relations
- **1** – use linear strain-displacement relations

IPLAS  
material-nonlinearity flag:

- **0** – elastic behavior only
- **1** – plasticity included, with the material law satisfied at each element integration point
- **2** – plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)

NX  
number of elements to be generated in the $x$ direction;

STAGS sets **NX = 1** if it is nonpositive

NY  
number of elements to be generated in the $y$ direction;

STAGS sets **NY = 1** if it is nonpositive

---

*go to T-882b*
**T-882a E882 Solid Element Nodes**

A single T-882a record must follow the T-882 record, to identify the 18 user nodes (S-1) that define the first solid element of the set to be generated here.

\[
\begin{align*}
\text{( NODE(k), } k = 1, 18 ) & \quad \text{USERELT} \\
\text{NODE(k)} & \quad k^{th} \text{ of 18 node points for the initial E882 solid element} \\
\text{USERELT} & \quad \text{user-specified element number, used only if } \text{IUWLE} = 1 \text{ on H-1} \\
\end{align*}
\]

\[
\begin{align*}
\text{NX} & \quad \text{(T-882) x-direction incrementation flag} \\
\text{NY} & \quad \text{(T-882) y-direction incrementation flag} \\
\text{IANG} & \quad \text{(T-882) wall-reference option} \\
\text{if} \quad ( \text{NX} > 1 ) & \quad \text{then go to T-882b} \\
\text{elseif} \quad ( \text{NY} > 1 ) & \quad \text{then go to T-882c} \\
\text{elseif} \quad ( \text{IANG} > 0 ) & \quad \text{then go to T-882d} \\
\text{else} & \quad \text{return to T-100}
\end{align*}
\]
T-882b E882 X-Direction Incrementations

Nineteen x-direction incrementation variables are specified here for use with the nx looping function invoked on the T-882 record and the initial node points established on T-882a.

( IX(k), k=1,18 ) IXU

IX(k) x-direction incrementation variable for node NODE(k) on T-882a
IXU x-direction incrementation variable for use with USERELT

Any of these incrementation variables can be negative, zero or positive.

if ( NY > 1 ) then go to T-882c
elseif ( IANG = 1 ) then go to T-882d
else return to T-100
T-882c E882 Y-Direction Incrementations

Nineteen y-direction incrementation variables are specified here for use with the NY looping function invoked on the T-882 record and the initial node points established on T-882a.

\[( IY(k), k=1,18 ) \quad IYU \]

<table>
<thead>
<tr>
<th>IY(k)</th>
<th>y-direction incrementation variable for node NODE(k) on T-882a</th>
</tr>
</thead>
<tbody>
<tr>
<td>IYU</td>
<td>y-direction incrementation variable for use with USERELT</td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero or positive.

if ( IANG = 1 ) then go to T-882d
else return to T-100
T-882d E882 Material Orientation Record

If \( IANG > 0 \) (T-882), a single T-882d record must follow the T-882a record (or T-882b or T-882c record, if applicable) for the current E882 solid element. T-882d specifies the material orientation for each of the E882 elements that are to be generated by the current T-882 element-definition set.

\[
\begin{align*}
XFX & \quad XFY & \quad XFZ \\
YFX & \quad YFY & \quad YFZ
\end{align*}
\]

- **XFX, XFY, XFZ** vector components establishing the x orientation of the material
- **YFX, YFY, YFZ** vector components establishing the y orientation of the material

\( \text{return to T-100} \)
**T-883 E883 27-Node Solid Element**

The **E883** 27-node ANS solid element, in STAGS, is shown in Figure 9.12:

![E883 27-Node ANS Solid Element Diagram](image)

Figure 9.12  E883 27-Node ANS Solid Element

The nodal ordering for **E883** corresponds exactly to the ordering used in the PATRAN program.

The T-883 record is the first record of a multi-record set that defines one or more **E883** 27-node solid elements to be included in the current element unit. The *looping* capabilities provided here, *via* `NX` and `NY` on T-883 and the incrementation parameters on T-883b and T-883c, can be used effectively in many situations.
### KELT  IFAB  IANG  ILIN  IPLAS  NX  NY

**KELT**
- must = 883

**IFAB**
- fabrication identifier for the element:
  - >0 – wall configuration number in the Wall Fabrication Table (K-1)
  - 0 – wall properties are given in user-written subroutine WALL
  - <0 – fabrication identifier in the GCP Fabrication Table (I-22a)

**IANG**
- wall-reference option:
  - 0 – use default strategy of projecting \(x_g, y_g\) to establish \(x_w\)
  - 1 – input \(r_w\), which is projected to establish \(x_w\)

**ILIN**
- geometric-nonlinearity flag:
  - 0 – use nonlinear strain-displacement relations
  - 1 – use linear strain-displacement relations

**IPLAS**
- material-nonlinearity flag:
  - 0 – elastic behavior only
  - 1 – plasticity included, with the material law satisfied at each element integration point
  - 2 – plasticity included, with the material law satisfied at the element centroid (centroidal plasticity)

**NX**
- number of elements to be generated in the \(x\) direction;
  - STAGS sets \(NX = 1\) if it is nonpositive

**NY**
- number of elements to be generated in the \(y\) direction;
  - STAGS sets \(NY = 1\) if it is nonpositive

---

\textit{go to T-883b}
**T-883a E883 Solid Element Nodes**

A single T-883a record must follow the T-883 record, to identify the 27 user nodes (S-1) that define the first solid element of the set to be generated here.

\[
(\text{NODE}(k), \ k=1,27 ) \quad \text{USERELT}
\]

**NODE(k)** $k^{th}$ of 27 node points for the initial E883 solid element

**USERELT** user-specified element number, used only if IUWLE = 1 on H-1

- **NX** (T-883) x-direction incrementation flag
- **NY** (T-883) y-direction incrementation flag
- **IANG** (T-883) wall-reference option

if (NX > 1) then go to T-883b
elseif (NY > 1) then go to T-883c
elseif (IANG > 0) then go to T-883d
else return to T-100
T-883b E883 X-Direction Incrementations

Twenty eight x-direction incrementation variables are specified here for use with the \textbf{nx} looping function invoked on the T-883 record and the initial node points established on T-883a.

\[( \text{IX}(k), k=1,27 ) \quad \text{IXU} \]

\begin{itemize}
  \item \textbf{IX}(k) \quad \text{x-direction incrementation variable for node NODE}(k) on T-883a
  \item \textbf{IXU} \quad \text{x-direction incrementation variable for use with USERELT}
\end{itemize}

Any of these incrementation variables can be negative, zero or positive.

\[\begin{align*}
\text{if} \quad ( \text{NY} > 1 ) & \quad \text{then go to T-883c} \\
\text{elseif} \quad ( \text{IANG} = 1 ) & \quad \text{then go to T-883d} \\
\text{else} & \quad \text{return to T-100}
\end{align*}\]
**T-883c E883 Y-Direction Incrementations**

Twenty eight y-direction incrementation variables are specified here for use with the NY looping function invoked on the T-883 record and the initial node points established on T-883a.

\[
\text{ ( IY(k), k=1,27 ) IYU }
\]

<table>
<thead>
<tr>
<th>IY(k)</th>
<th>y-direction incrementation variable for node NODE(k) on T-883a</th>
</tr>
</thead>
<tbody>
<tr>
<td>IYU</td>
<td>y-direction incrementation variable for use with USERELT</td>
</tr>
</tbody>
</table>

Any of these incrementation variables can be negative, zero or positive.

\[
\text{ if ( IANG = 1 ) then go to T-883d}
\]

else return to T-100
T-883d E883 Material Orientation Record

If $\text{iang} > 0$ (T-883), a single T-883d record must follow the T-883a record (or T-883b or T-883c record, if applicable) for the current E883 solid element. T-883d specifies the material orientation for each of the E883 elements that are to be generated by the current T-883 element-definition set.

\[
\begin{array}{ccc}
XFX & XFY & XFZ \\
YFX & YFY & YFZ
\end{array}
\]

- $XFX$, $XYF$, $XFZ$ vector components establishing the x orientation of the material
- $YFX$, $YFY$, $YFZ$ vector components establishing the y orientation of the material

\[\text{return to } \text{T-100}\]
**T-885 E885 20-Node Solid Element**

The **E885** 20-node displacement-based solid element, in **STAGS**, is shown in Figure 9.13:

![Figure 9.13 E885 20-Node Solid Element](image)

The nodal ordering for **E885** corresponds exactly to the ordering used in the **PATRAN** program.

The T-885 record is the first record of a multi-record set that defines one or more **E885** 20-node solid elements to be included in the current element unit. The *looping* capabilities provided here, via **NX** and **NY** on T-885 and the incrementation parameters on T-885b and T-885c, can be used effectively in many situations.
<table>
<thead>
<tr>
<th><strong>KELT</strong></th>
<th><strong>IFAB</strong></th>
<th><strong>IANG</strong></th>
<th><strong>ILIN</strong></th>
<th><strong>IPLAS</strong></th>
<th><strong>NX</strong></th>
<th><strong>NY</strong></th>
</tr>
</thead>
</table>
| must = 885 | fabrication identifier for the element; this is equivalent to the WALL parameter for E410 quadrilateral elements:  
  >0 – wall configuration number in the Wall Fabrication Table (K-1)  
  0 – wall properties are given in user-written subroutine WALL  
  <0 – fabrication identifier in the GCP Fabrication Table (I-22a) | wall-reference option:  
  0 – use default strategy of projecting \( x_f, y_g \) to establish \( x_w \)  
  1 – input \( r_w \), which is projected to establish \( x_w \) | geometric-nonlinearity flag:  
  0 – use nonlinear strain-displacement relations  
  1 – use linear strain-displacement relations | material-nonlinearity flag:  
  0 – elastic behavior only  
  1 – plasticity included, with the material law satisfied at each element integration point  
  2 – plasticity included, with the material law satisfied at the element centroid (centroidal plasticity) | number of elements to be generated in the x direction;  
STAGS sets **NX** = 1 if it is nonpositive | number of elements to be generated in the y direction;  
STAGS sets **NY** = 1 if it is nonpositive |

\[ \text{go to T-885b} \]
**T-885a E885 Solid Element Nodes**

A single T-885a record must follow the T-885 record, to identify the 20 user nodes (S-1) that define the first solid element of the set to be generated here.

\[
( \text{NODE}(k), \; k=1,20 ) \quad \text{USERELT}
\]

**NODE(k)** \(k^{th}\) of 20 node points for the initial E885 solid element

**USERELT** user-specified element number, used only if \(\text{IUWLE} = 1\) on H-1

\[
\begin{align*}
\text{NX} & \quad \text{(T-885)} \quad \text{x-direction incrementation flag} \\
\text{NY} & \quad \text{(T-885)} \quad \text{y-direction incrementation flag} \\
\text{IANG} & \quad \text{(T-885)} \quad \text{wall-reference option} \\
\text{if} & \quad ( \text{NX} > 1 ) \quad \text{then go to T-885b} \\
\text{elseif} & \quad ( \text{NY} > 1 ) \quad \text{then go to T-885c} \\
\text{elseif} & \quad ( \text{IANG} > 0 ) \quad \text{then go to T-885d} \\
\text{else} & \quad \text{return to T-100}
\end{align*}
\]
T-885b E885 X-Direction Incrementations

Twenty one x-direction incrementation variables are specified here for use with the \( \text{nx} \) looping function invoked on the T-885 record and the initial node points established on T-885a.

\[
(\text{IX}(k), \ k=1,\ldots,20) \quad \text{IXU}
\]

\(\text{IX}(k)\) \hspace{2cm} x-direction incrementation variable for node \(\text{NODE}(k)\) on T-885a
\(\text{IXU}\) \hspace{2cm} x-direction incrementation variable for use with \(\text{USERELT}\)

Any of these incrementation variables can be negative, zero or positive.

\textcolor{red}{\begin{align*}
\text{if} \quad (\text{NY} > 1) \quad \text{then} \quad & \text{go to T-885c} \\
\text{elseif} \quad (\text{IANG} = 1) \quad \text{then} \quad & \text{go to T-885d} \\
\text{else} \quad & \text{return to T-100}
\end{align*}}\}
T-885c E885 Y-Direction Incrementations

Twenty one y-direction incrementation variables are specified here for use with the NY looping function invoked on the T-885 record and the initial node points established on T-885a.

\[(I_y(k), k=1,20) \quad I_yU\]

- \(I_y(k)\) y-direction incrementation variable for node \(NODE(k)\) on T-885a
- \(I_yU\) y-direction incrementation variable for use with USERELT

Any of these incrementation variables can be negative, zero or positive.

if ( \(IANG = 1\) ) then \(go \ to \ T-885d\)
else \(return \ to \ T-100\)
T-885d E885 Material Orientation Record

If $\text{IANG} > 0$ (T-885), a single T-885d record must follow the T-885a record (or T-885b or T-885c record, if applicable) for the current E885 solid element. T-885d specifies the material orientation for each of the E885 elements that are to be generated by the current T-885 element-definition set.

\[
\begin{align*}
XFX & \quad XFY & \quad XFZ \\
YFX & \quad YFY & \quad YFZ
\end{align*}
\]

- $XFX, XFY, XFZ$ vector components establishing the x orientation of the material
- $YFX, YFY, YFZ$ vector components establishing the y orientation of the material

return to T-100


9.8 Definition and Utilization of User Elements

Recently-developed User element capabilities in STAGS 5.0 give advanced analysts and program developers powerful new tools for solving problems that lie beyond the scope of those that STAGS has traditionally treated. The process by which User elements are defined in STAGS is described briefly in the following Subsection, in somewhat greater detail in Chapter 13 of this document, and much more fully in the STAGS Elements Manual. The basic input records with which User elements may be utilized in constructing a STAGS model are described after that.

9.8.1 The User–element definition process

There are two distinct (but strongly coupled) operations that must be performed to define a User element for use in the construction of STAGS models and for use in conducting analyses with those models.

The first operation (which must be performed in generating the INP file that STAGS’s s1 processor reads and processes to construct the STAGS model for a given problem) is the analyst’s definition of certain critical information about the User element—the element–type identifier by which it is to be referenced, the number of node points that are required to specify any given element of this type, the names and characteristics of data associated with all elements of this type, etc. This is easily accomplished by including a set of User-element-definition directives at any appropriate point in the analyst’s INP file prior to any input records that command the inclusion of one or more User elements of that type in the model being constructed.

The second operation (which must generally be performed prior to executing any STAGS processor) is the analyst’s (or developer’s) generation of FORTRAN and/or C–language routines that STAGS processors must employ in the specification, analysis and postprocessing operations that are to be performed for any STAGS model in which these User elements are employed—and the successful linking of each of these User-element-enhanced processors.

Before getting into either of these operations, it is appropriate to pause briefly and note that the principal goal of the STAGS User element framework is to provide a set of features that enable element researchers to add new element types to STAGS without having to be concerned with details of the STAGS element integration process—focusing only on the development of their new elements. To provide an almost transparent element integration environment, the STAGS User element framework provides the following features:
- Arbitrary User element types are represented in the standard STAGS element description scheme as \(300 \leq \text{type} \leq 999\).
- Any number (less than or equal to one hundred) of User element types and their associated data descriptions may be specified.
- Any number of User elements of any User element type may be defined in STAGS models.
- Each User element type definition specifies a set of named data that are available for every element of that type.
- User element program code can retrieve and store the values of individual User element data items by their names, or can access an element's integer and floating-point data blocks in the ABAQUS style.
- An arbitrary number of user-described Property Sets can be defined that specify a collection of named data that may be arbitrarily associated with one or more elements of any user-defined type.
- User element wrapper routines are invoked automatically by STAGS as required by a solution process. Users implementing new elements need not be concerned with the details of the STAGS solution process.

An arbitrary number of user-defined element types can be specified using a free field description of the nodal and data specifications for each element. These descriptions are entered directly into the STAGS model input file and provide the user the ability to describe the element type and its data requirements in any manner that facilitates the most natural ordering. Comments and flexible line spacing are allowed for enhanced readability.

Each User element type description may specify a collection of named integer and/or floating-point variables associated with each User element defined in a STAGS model. Naming User element variables greatly increases a user's understanding of both simple and complex datasets and reduces the likelihood of confusing data values required by different element types. This is a general capability; the developer or researcher is not restricted by the kind or amount of data that can be associated with each User element type.

The User element framework also introduces a new STAGS feature called Property Sets. A developer or researcher may define any number of property sets, each of which is identified by a meaningful name and a unique numeric identifier. The property items in a property set, entered by the user in free field format (much like User element type descriptions), collect integer and/or floating-point data that are best described when grouped together. This feature provides a general and flexible technique for describing data required by any new capabilities added by developers or researchers. The creation of new, user-defined element types is just one example.
Directives for User element definitions

Returning to the subject of *directives* for definition of User elements, we note that each User element type is described in the *INP* file for a *STAGS* model by a set of four required directives and two optional directives (within square brackets)—that are ordered as follows:

*userElement... — initiate userElement specifications
   *dofOrdering... — specify number and types of DOF at each node
   *nodeSequence... — specify nodal sequence for transformations
   [*floatVariables...] — specify float-type variables (if any)
   [*integerVariables...] — specify integer-type variables (if any)
*end userElement — terminate userElement specifications

Directive formatting is free-field. String values that contain one or more spaces must be enclosed in matching single or double quotes. Directive names and directive attribute names are case-insensitive. Empty (blank) lines and lines containing only $-initiated comments are ignored during parsing of directives.

The *floatVariables* and *integerVariables* data groups are both optional. They may appear in any order and may be repeated as many times as required. All float variable items are collected sequentially as they are defined and are placed in a single logical group of data type float. All integer variable items are collected sequentially as they are defined and are placed in a single logical group of type integer.

Example of User element definition

Rather than going through a long-winded presentation of directive syntax and conventions, let us simplify this by looking at a straightforward example that shows how a user might define a User element—to be referenced as a type 901 element—for a simple 3-node beam. The end nodes of this beam are identified as node number 1 and node number 2, and the internal node is identified as node number 3. A fourth node is associated with the element in order to compute the element’s orientation matrix. Associated with this element type is a set of named floating-point variables and a set of named integer variables. These named data will be defined for each user-defined element created by *STAGS* and can be manipulated by developers and by the element researcher using utility routines that are described in Chapter 13 of this document and in greater detail in Chapter 9 of the *STAGS Elements Manual*. 
The following points about this example are generally true for all User element type definitions.

- User element type names must have string values and are used for descriptive purposes only.
- User element type names are limited to 40 characters in length.
- User element type numbers must be in the range $900 \leq \text{type} \leq 999$, inclusive; and they must be unique (i.e., a User element type must be defined only once).
- Some of the type numbers in the $900 \leq \text{type} \leq 999$ cannot be used for new User elements because they are reserved for (already being used by) “built-in” STAGS UELs (by the \textbf{E928} and \textbf{E940} elements, for example).
- A User element type may have any number of nodes.
- The DOF ordering for each node must be specified in a single group.
- All nodes are expected to have either zero (0), three (3) or six (6) degrees of freedom. DOF must currently be defined in agreement with STAGS’ DOF-ordering convention:
  
  $\begin{align*}
  1 & : \text{x–displacement} \\
  2 & : \text{y–displacement} \\
  3 & : \text{z–displacement} \\
  4 & : \text{x–rotation} \\
  5 & : \text{y–rotation} \\
  6 & : \text{z–rotation}
  \end{align*}$

Element reference nodes are specified as having zero degrees of freedom (as for node 4 in this example).

Nodes used to define an element’s orientation matrix are identified by a single, required *nodeSequence directive. Either three (3) or four (4) nodes in the *dofOrdering group are referenced by index in this directive. If three nodes are referenced, the element’s x-axis is directed from the first to the second node. The element’s z-axis is directed as the vector cross product of the x-axis and the vector from the first to the third node. The element’s y-axis follows from the right-hand rule with respect to the x-axis and the z-axis. If four nodes are referenced, the element’s x-axis is directed from the first to the second node. The element’s z-axis is directed as the vector cross product of the vector from the first to the third node and the vector from the second to the fourth node. The element’s y-axis follows from the right-hand rule with respect to the x-axis and the z-axis.

User element variable names are limited to 40 characters in length.

Variable names containing one or more spaces must be enclosed in matching single or double quotes.

Variable names are case-insensitive for comparison and must be unique within their User element type definition.

A variable's size (logical length) must be specified and must be an integer number greater than zero.

**Directives for User property set definitions**

User property sets are described (during STAGS sI processing) through a set of directives as shown in the following example. Directives defining a User property set are ordered as follows:

*userProperty... – initiate userProperty specifications

[*floatProps...] – define one or more float-type data parameters

[*integerProps...] – define one or more integer-type data parameters

*end userProperty – initiate userProperty specifications

As with the User-element-definition directives described above, empty (blank) lines and lines containing only $-initiated comments are ignored during parsing of directives; and directive names and directive attribute names are case-insensitive. Directive formatting is free-field. String values that contain one or more spaces must be enclosed in matching single or double quotes.

The *floatProps data group is optional, and *floatProps data groups may appear in any order and may be repeated as many times as required. All float property items are collected sequentially as they are defined and are placed in a single logical group of data type float.
The *integerProps data group is optional when the analyst/developer chooses not to use STAGS’ GCP processor to specify material properties and fabrications; it is mandatory when he/she uses the GCP to do that. When needed, *integerProps data groups may appear in any order and may be repeated as many times as required. All integer property items are collected sequentially as they are defined and are placed in a single logical group of data type integer. Parameters that must be specified in the *integerProps data group when the GCP is used are described briefly in Chapter 13 of this document and more completely in Chapter 9 of the STAGS Elements Manual.

Examples of User property set definitions

These examples build on the previous example that showed how an element researcher might define a new element type for a 3-node beam with a single internal node. In addition to the element type’s set of named integer and floating-point element variables, there may be several User property sets defined to complete the specification of an element. These property sets contain collections of related property items and can be logically associated with any number of elements or referenced from any number of appropriate contexts. User property sets can be manipulated by developers and element researchers using utility routines that are described briefly in Chapter 13 of this document and more completely in Chapter 9 of the STAGS Elements Manual.

```
*userProperty  name = "Beam Element"  id = 901
   *integerProps
      $  Required Standard Data
      $  ----------------------
      ActiveNodes     2
      SamplingCount  10
      StrainCount     6
      StressCount     6
   *end userProperty

*userProperty  name = "Aluminum 6061-T6"  id = 6061
   *floatProps
      E                        10.00e+6
      G                         3.75e+6
      MassDensity               2.54e-4
      PoissonRatio              0.3
      TensileUltimateStrength  38.00e+3
      TensileYieldStrength     35.00e+3
      CompressiveYieldStrength 35.00e+3
      ShearYieldStrength       20.00e+3
      ThermalExpansion         1.30e-5
   *end userProperty
```
Notes for User property set definition examples

The following points made about these examples are generally true for all User property set definitions:

- For each User element type definition there must be one property set with an identifier (id) that matches the identifier of the type definition. In addition to any other properties this property set may contain, the following integer property items must be specified: ActiveNodes (the number of active nodes for the element type), SamplingCount (the number of locations where stresses and strains will be evaluated), StrainCount (the number of strain components associated with the element type), StressCount (the number of stress components associated with the element type). Failure to provide these property items in a property set associated with an element type definition will lead to a run-time error.

- User property set names must have string values and must be unique across all User property sets.

- Property set names are limited to 40 characters in length.

- User property set identifiers (IDs) must have numeric values and must be unique across all User property sets.

- Property item names containing one or more spaces must be enclosed in matching single or double quotes

- Property item names are limited to 40 characters in length.

- Property item names are case-insensitive for comparison and must be unique within their property set definition.

Subroutines facilitating User element implementations

When one or more types of User elements are to be employed in a STAGS model, it is sometimes necessary for the analyst (or developer) to generate a User-element-enhanced version of STAGS’ s1 (model-definition) processor. It is always necessary for him (or her) to generate User-element-enhanced versions of STAGS’ s2 (analysis) processor and of any of the STAGS–system post–analysis processor(s) that are to be used with that model. The broad outlines of how this may be done are sketched in the following paragraphs. The interested reader should consult Chapter 13 of this document for more information and should see Chapter 9 of the STAGS Elements Manual for all of the gory details.
It is sufficient to note here that when STAGS’ s1 processor encounters a User element, s1 calls a number of generic User-element definition routines to perform operations that are required universally; and it also calls three other User-element “dispatch” routines to perform additional, nonstandard model-definition operations for that User element—as and if they are required. The primary function of each of these dispatch routines is evident by its name: UelEltDef, UelLoadDef and UelPressDef. Each of these dispatch routines calls one or more next-level program- or developer-supplied routines that perform nonstandard definition operations that are appropriate for each type of User element. The “program-supplied” qualifier is appropriate when no special operations are required; the “developer-supplied” qualifier applies when special operations are required. In any event, the program “architecture” within which this is done is shown in the following Table:

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>calls subroutine</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>UelEltDef</td>
<td>UelEltDef900</td>
<td>for a type 900 element</td>
</tr>
<tr>
<td>UelEltDef</td>
<td>UelEltDef901</td>
<td>for a type 901 element</td>
</tr>
<tr>
<td>UelEltDef</td>
<td>UelEltDef902</td>
<td>for a type 902 element</td>
</tr>
<tr>
<td>UelLoadDef</td>
<td>UelLoadDef900</td>
<td>for a type 900 element</td>
</tr>
<tr>
<td>UelLoadDef</td>
<td>UelLoadDef901</td>
<td>for a type 901 element</td>
</tr>
<tr>
<td>UelLoadDef</td>
<td>UelLoadDef902</td>
<td>for a type 902 element</td>
</tr>
<tr>
<td>UelPressDef</td>
<td>UelPressDef900</td>
<td>for a type 900 element</td>
</tr>
<tr>
<td>UelPressDef</td>
<td>UelPressDef901</td>
<td>for a type 901 element</td>
</tr>
<tr>
<td>UelPressDef</td>
<td>UelPressDef902</td>
<td>for a type 902 element</td>
</tr>
</tbody>
</table>

The names of these nine User-element-type-specific subroutines (for hypothetical user types 900, 901 and 902) are arbitrary (within the constraint that each unique User element type number must lie in the range from 900 to 999, inclusive) and may be changed to reflect the user’s particular choice. A do-nothing “starter” template is provided for each of these particular routines. The three s1-processor dispatch routines and their type-specific children are described more fully in Chapter 13 of this document and are fully documented in Chapter 9 of the STAGS Elements Manual. A comprehensive set of FORTRAN and C-language utility routines that is provided with STAGS to facilitate their implementation and use in s1 and other STAGS processors is also documented there.

Similarly, when the STAGS’ s2 processes encounters a User element, it also calls a number of User-element “dispatch” routines to perform initialization and computation operations for that User element—as required. Depending on the type of analysis to be performed, s2 may call some or all of the twelve User-element dispatch routines that are listed in the following Table:
Each of these top-level dispatch routines in turn calls a corresponding lower-level, user-defined routine that is specific to a particular user-defined element type. For example, \texttt{UelPvDef} calls the provided templates \texttt{UelPvDef900}, \texttt{UelPvDef901} and \texttt{UelPvDef902}. Similar template routines are provided for the other dispatch routines. The names of these user element type-specific routines (for hypothetical user types 900, 901 and 902) are arbitrary and may be changed to reflect the user’s particular choice. These \texttt{s2}-processor dispatch routines and their User-element-type-specific children are described in Chapter 13 of this document and are fully documented in Chapter 9 of the \textit{STAGS Elements Manual}.

\subsection*{9.8.2 Utilization of User element(s) in STAGS}

The basic input requirements for User elements are similar to those for other elements in the \textit{STAGS} program. The reader can verify this by studying the following pages carefully.
**T-900 E9XX User-Defined Element**

In the current version of STAGS, the T-900 record is used to identify the type of UEL that the user wants to add to his current element unit (via the KELT variable) and to specify the values of of eight other variables that STAGS uses to specify one or more UELs of that type. In the “general” case, the user can specify any type of UEL here that the program “recognizes” and “understands.” This description of T-900 is intended to be used by developers and analysts who want to use STAGS’ UEL-definition framework to develop, test, and use new types of User Written elements in and with the program. A STAGS developer/user who is doing that will typically be dealing with a new element that looks something like this

![Figure 9.14 E9XX User-Written Element](image)

where the black dots are nodes, where the element is defined in an element coordinate system that looks something like the one shown here, and where the developer/user’s element typically looks like something other than a potato shown here. The developer or user who wants to develop or use this element in STAGS (as a User-Written element) will typically assign it a UEL type code (KELT identity) that lies somewhere in the range \(900 \leq \text{type} \leq 999\). He should not use \(910\) or \(928\) or \(940\) as the type code for his UEL because those three type codes are not available to him. STAGS uses those three type codes for User Written elements that have been implemented in the program as “built-in” UELs that anyone can use—the program’s type-910 “actuator” element, its type-928 “3-node curved beam” element, and its type-940 “4-node MIN4 quadrilateral” element.

The user who wants to add one or more of STAGS’ “built-in” UELs to his current element unit should stop reading this description of the T-900 record and turn his attention to its E910, E928, and/or E940 counterparts later in this section.
The user who is developing or using other types of STAGS UELs should know that, whatever unique input requirements there may be for specifying one or more UELs of any given type, he must first specify the most basic things that STAGS needs to know about that type of UEL—by including the appropriate type-definition directives for it in his case.inp file at an appropriate point before he adds any element of that type to any element unit. The user must also specify the (floating point and/or integer) values that he wants STAGS to use for properties and/or control variables in any property sets that his UEL uses—by including the appropriate property-set directive(s) in his case.inp file somewhere before STAGS needs that information.

Then... finally... the user must navigate to and use the T-900 record described here (a) to specify the type of UEL that he wants to add to his current element unit, (b) to specify the “element number” of the first UEL of this type to be added at this time, (c) to specify four input and control flags, and to specify three incrementation variables—via the KELT, ID, IFAB, IANG, ILIN, IPLAS, NX, NY and NZ variables on the following record:
Note 1: If $KELT = 910$ on this T-900 record, the contents of this record are more accurately described on the T-910 record that is discussed later in this section.

If $KELT = 928$ on this T-900 record, the contents of this record are more accurately described on the T-928 record that is discussed later in this section.

If $KELT = 940$ on this T-900 record, the contents of this record are more accurately described on the T-940 record that is discussed later in this section.

Note 2: The current version of STAGS’ E928 3-node curved beam element—its “built-in” type-928 UEL—“ignores” the IFAB, IANG, ILIN and IPLAS variables on this record because that element gets all of its fabrication and material property data from UEL directives; when $KELT=928$ on this record, the user must currently set each of these “place-holder” variables equal to 0 — at least until STAGS’ E928 UEL is upgraded to use the program’s more flexible GCP facilities.

The current version of STAGS’ E940 MIN4 quadrilateral element—its “built-in” type-940 UEL—uses STAGS’ GCP facilities exclusively (and does not use its older fabrication and material specification and computation methodologies); when $KELT=940$ on this record, the user must set IFAB equal to a negative number (in accord with the GCP fabrication that the current E940 elements use), and he must also set IANG=ILIN=IPLAS=0 (for the time being, at least).

Note 3: The NX variable on this record can be used (with most types of UELs) to generate a “string” of elements in the string’s “row” space; the NX and NY variables can be used together (with many types of UELs) to generate a “patch” of elements in its “row/column” space; and the NX, NY and NZ variables can be used (with a few types of UELs) to generate a “block” of elements in its “row/column/layer” space.

The current version of STAGS’ E928 element “ignores” the NY and NZ variables on this record; when $KELT=928$ on this record, the user should set NY=NZ=0 (or 1).

The current version of STAGS’ E940 element “ignores” the NZ variable on this record; when $KELT=940$ on this record, the user should set NZ=0 (or 1).

```plaintext
if ( KELT = 928 ) then
  go to T-928a
elseif ( KELT = 940 ) then
  go to T-940a
else
  go to T-900a
endif
```
T-900a E9XX User-Element Nodes

For most types of UELs, a single T-900a record must follow the T-900 record—to identify the user nodes (S-1 or S-3) that define the first element of the set to be generated here. Type-specific equivalents to this record are described later in this section for “built-in” UELs: those records should be used instead of T-900a for “built-in” UELs.

(NODE(k), k = 1, NNODES)

NODE(k) the \( k^{th} \) of NNODES node points for the initial type-KELT UEL, where NNODES is specified by the *UserElement directive with which a type KELT User element is defined.

NX (T-900) x-direction incrementation flag
NY (T-900) y-direction incrementation flag
NZ (T-900) z-direction incrementation flag
IANG (T-900) wall-reference option

if (NX > 1) then
  go to T-900b
elseif (NY > 1) then
  go to T-900c
elseif (NZ > 1) then
  go to T-900d
elseif (IANG = 1) then
  go to T-900e
elseif (IANG = 2) then
  go to T-900f
else
  follow the instructions at the end of T-900f
endif
**T-900b User-Element X-Direction Incrementations**

**NNODES** incrementation variables are specified here for use with the **NX** (x-direction) looping function invoked on the T-900 record and the initial node points established on T-900a.

```
( IX(k), k = 1, NNODES ) IXU
```

<table>
<thead>
<tr>
<th>IX(k)</th>
<th>x-direction incrementation variable for node NODE(k) on T-900a</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXU</td>
<td>x-direction incrementation variable for use with the ID parameter on T-900a</td>
</tr>
</tbody>
</table>

Any of the **IX(k)** incrementation variables can be negative, zero or positive. **IXU** must be nonzero.

- **NY** (T-900) y-direction incrementation flag
- **NZ** (T-900) z-direction incrementation flag
- **IANG** (T-900) wall-reference option

if ( NY > 1 ) then
  go to T-900c
elseif ( NZ > 1 ) then
  go to T-900d
elseif ( IANG = 1 ) then
  go to T-900e
elseif ( IANG = 2 ) then
  go to T-900f
else
  follow the instructions at the end of T-900f
endif
**T-900c User-Element Y-Direction Incrementations**

**NNODES** incrementation variables are specified here for use with the **NY** (y-direction) looping function invoked on the T-900 record and the initial node points established on T-900a.

\[
( \text{IY}(k), k = 1, \text{NNODES} ) \quad \text{IYU}
\]

**IY(k)**  
y-direction incrementation variable for node **NODE(k)** on T-900a

**IYU**  
y-direction incrementation variable for use with the **ID** parameter on T-900a

Any of the **IY(k)** incrementation variables can be negative, zero or positive. **IYU** must be nonzero.

\[
\text{NZ} \quad (\text{T-900}) \quad \text{z-direction incrementation flag}
\]

\[
\text{IANG} \quad (\text{T-900}) \quad \text{wall-reference option}
\]

if  
( **NZ** > 1 )  
then

go to **T-900d**
elseif  
( **IANG** = 1 )  
then

go to **T-900e**
elseif  
( **IANG** = 2 )  
then

go to **T-900f**
else

follow the instructions at the end of **T-900f**
endif
**T-900d User-Element Z-Direction Incrementations**

**NNODES** incrementation variables are specified here for use with the **NZ** (z-direction) looping function invoked on the T-900 record and the initial node points established on T-900a.

\[
(I_Z(k), \ k = 1, \ \text{NNODES}) \quad I_ZU
\]

- **IZ(k)**: z-direction incrementation variable for node **NODE(k)** on T-900a
- **IZU**: z-direction incrementation variable for use with the **ID** parameter on T-900a

Any of the **IZ(k)** incrementation variables can be negative, zero or positive. **IZU** must be nonzero.

**IANG** (T-900) wall-reference option

if \( IANG = 1 \) then
  go to T-900e
elseif \( IANG = 2 \) then
  go to T-900f
else
  follow the instructions at the end of T-900f
endif
T-900e User-Element Material Orientation Vector

If $IANG = 1$ (T-900), a single T-900e record must follow T-900a (or T-900b or T-900c or T-900d) for the current set of User elements. T-900e specifies a single x-direction vector that is used to establish the material orientation triad for each of the User elements that are to be generated by the current T-900 element-definition set.

\begin{align*}
  XFX & \quad XFY & \quad XFZ \\
\end{align*}

$XFX, XFY, XFZ$ vector components establishing the x orientation of the material

follow the instructions at the end of T-900f
**T-900f User-Element Material Orientation Vectors**

If \( \text{IANG} = 2 \) (T-900), a single T-900f record must follow T-900a (or T-900b or T-900c or T-900d) for the current set of User elements. T-900f specifies an \( x \)-direction and a \( y \)-direction vector that are both used to establish the material orientation triad for each of the User elements that are to be generated by the current T-900 element-definition set.

\[
\begin{align*}
\text{XFX} & \quad \text{XFY} & \quad \text{XFZ} & \quad \text{YFX} & \quad \text{YFY} & \quad \text{YFZ} \\
\text{XFX, XFY, XFZ} & \quad \text{vector components establishing the x orientation of the material} \\
\text{YFX, YFY, YFZ} & \quad \text{vector components establishing the y orientation of the material}
\end{align*}
\]

```plaintext
if  ( the user wants to add another set of \textbf{E900} elements )  then
go to \textbf{T-900}
elseif ( the user wants to add a set of \textbf{E928} elements )  then
go to \textbf{T-928}
elseif ( the user wants to add a set of \textbf{E940} elements )  then
go to \textbf{T-940}
else
  return to \textbf{T-100}
endif
```
**T-928 E928 3-Node Curved Beam UEL**

STAGS’ E928 3-node curved beam element—shown in the following Figure—is fully compatible with its classic E480 9-node ANS quadrilateral shell element and its E849 18-node sandwich element (which uses E480 elements for its top and bottom face sheets). STAGS’ E928 element was originally implemented in the program as a User-Defined Element, in accord with the UEL specifications and conventions that are described here and in Chapter 13 of this document. After testing this User-Defined element successfully, it was incorporated into STAGS as a “built-in” UEL that anyone can use without having to re-make s1, s2 and other STAGS processors to do so.

![Figure 9.15 E928 3-Node Curved Beam Element](image)

This T-928 element-specification record (set) should be used instead of the T-900 record (set) described earlier in this section when the user wants to add one or more “built-in” type-928 UELs to his current element unit. STAGS uses the same subroutines—and the same data spaces—to read and process all of the “generic” and type-specific input records that are needed for any type of UEL, so what we are calling the T-928 record here is actually a T-900 record that should be used as described here.
Before we turn our attentions to this record (and the other records in the T-928 record set), we need to remind the user whose model has one or more E928 UELs in it that he must include the following *userElement and *userProperty UEL-definition directives somewhere in his case.inp file for that model prior to his first use of any E928 element:

*userElement  name = “Uniform Beam Element”  type = 928  nodes = 4
*dofOrdering
$ Node DOF
$ ---- --- ---
1   1 2 3 4 5 6
2   1 2 3 4 5 6
3   1 2 3 4 5 6
4   0
*nodeSequence
$ Nodes...
$ --------
1   3   4   2
*floatVariables
$ Name Size
$ ---- ----
Area 1
Iy  1
Iz  1
J   1
Material 1
ShearFactorY 1
ShearFactorZ 1
Ecc  2
Scc  2
*end userElement
*userProperty  name = “Standard Data -- Uniform Beam Element”  id = 928
*integerProps
$ Required Standard Data
$ ------------------------
ActiveNodes  3
SamplingCount  10
StrainCount   6
StressCount   6
Beam          1
*floatProps
$ Nodal Stress/Strain Sampling Points
$ -----------------------------------
y1   -0.075
z1   -0.500
y2   0.075
z2   -0.500
y3   0.075
z3   0.500
y4   -0.075
z4   0.500
y5   0
z5   0
*end userProperty
These directives replicate the “standard” element-type properties that the developer of this UEL specified when he implemented his 3-node curved beam element in STAGS as a type-928 element. These element-definition directives are printed in Blue to remind the user that they must be included in his case.inp file “precisely” as they are shown here—with no substantive changes.

When the user’s model has one or more E928 elements in it, he must also include at least one *userProperty directive like the one that follows this paragraph—to specify the physical properties of each of the different materials that those elements use. Each of these *userProperty directives must have a unique name and a unique id, and each of them must be included in the user’s case.inp file prior to its first use by any UEL of any type.

*userProperty  name = “Aluminum 6061-T6”  id = 6061
*floatProps
  E      11.00000e+6
  G      4.13534e+6
  MassDensity  2.540e-4
  PoissonRatio  0.33
  TensileUltimateStrength  38.0e+3
  TensileYieldStrength  35.0e+3
  CompressiveYieldStrength  35.0e+3
  ShearYieldStrength  20.0e+3
  ThermalExpansion  1.3e-5
*end userProperty

The Blue items in this directive define “standard” parts of any E928 material specification data set and should be included “as is” in the analyst’s case.inp file. Each of the Red entries in this directive defines the value that the user specifies for the variable that precedes it — a short character string for the name variable, a (unique) integer value for the property identification variable (id), or a floating point value for one of the material-property variables (E = 11.0e+6, for example). The user can and should change the values of the Red entries whenever he needs to do so to construct the model that he needs for the case at hand. In many “simple” cases, all of the E928 elements in the user’s model use the same material properties: and he only needs to put one directive like this in his case.inp file. In more complex cases, where the model has E928 elements that use different materials from group to group or from element to element, he will need to define each different material via its own material-property-definition directive—before using that material in any UEL.

In any event... with a case.inp that has all of the directives in it that are needed to define what a type-928 element is and to specify the material properties that it needs, the user can (finally) use the T-928 record that is described below (a) to specify his intention to add one or more E928 elements to the current element unit, (b) to specify the “element number” of the first element in this set of elements, to specify “place-holder” values for the four input and control flags, and to
specify one incrementation variable—*via* the `KELT`, `ID`, `IFAB`, `IANG`, `ILIN`, `IPLAS` and `NX` variables on this T-928 record:

<table>
<thead>
<tr>
<th>KELT</th>
<th>ID</th>
<th>IFAB</th>
<th>IANG</th>
<th>ILIN</th>
<th>IPLAS</th>
<th>NX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **KELT** must be 928 for this **E928 UEL**. See **Note 1**, below.
- **ID** is the starting element number for the first element in this element set.
- **IFAB** must be 0 (fabrication identifier). See **Note 2**, below.
- **IANG** must be 0 (wall-reference option). See **Note 2**, below.
- **ILIN** must be 0 (geometric-nonlinearity flag). See **Note 2**, below.
- **IPLAS** must be 0 (material-nonlinearity flag). See **Note 2**, below.
- **NX** is the number of elements to be generated in the x direction (default = 1). See **Note 3**, below.

**Note 1**: Any **KELT** value other than 928 on this record tells STAGS that the user wants to stop adding **E928** elements and start adding one or more type-**KELT** UELs to the current element unit. In this case, the user should follow the prescriptions given elsewhere in this section for that type of UEL (instead of following the prescription given here for type-928 elements).

**Note 2**: **E928** elements get all of the fabrication and material property information that they can use from UEL directives (currently), and they do not use **IFAB**, **IANG**, **ILIN** or **IPLAS**; the prudent user should set each of these variables equal to 0—especially if he uses the **NX** variable to generate a “string” of **E928** UELs.

**Note 3**: The user can generate a string of **E928** UELs using the **NX** incrementation variable on this record, but he cannot (currently) generate 2- or 3-dimensional sets of strings by using the **NY** and **NZ** parameters that T-900 accommodates; the prudent user should refrain from using T-900’s **NY** and **NZ** variables—or set both of them equal to 1 if he cannot bear to do that.

*go to T-928a*
### T-928a E928 UEL Nodes

This is where the user specifies the four node points that define the first E928 element that he wants to add to the current element unit via the current T-928 record set. STAGS “knows” that he must specify four nodes here because the *userElement* directive that defines what a type-928 element “is” specified that each type-928 element in any given element unit must have four nodes—each of which is a node that is included in the nodal point list that he constructed via his S-1 or S-3 specifications for that unit.

Each T-928 record must be followed by a single T-928a E928 UEL NODES record:

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>node # 1, at end “A” of the beam</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>node # 2, at the “center” of the beam</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>node # 3, at end “B” of the beam</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N4</td>
<td>reference node</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NX (T-928) x-direction incrementation flag

if (NX > 1) then
go to T-928b
else
go to T-928c
endif
**T-928b E928 UEL X-Direction Incrementations**

Incrementation variables are specified here for use with the **nx** (x-direction) looping function invoked on the **T-928** record and the initial node points established on **T-928a**.

<table>
<thead>
<tr>
<th>IX1</th>
<th>IX2</th>
<th>IX3</th>
<th>IX4</th>
<th>IXU</th>
</tr>
</thead>
<tbody>
<tr>
<td>IX1</td>
<td>x-direction incrementation variable for node <strong>N1</strong> on <strong>T-928a</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX2</td>
<td>x-direction incrementation variable for node <strong>N2</strong> on <strong>T-928a</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX3</td>
<td>x-direction incrementation variable for node <strong>N3</strong> on <strong>T-928a</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX4</td>
<td>x-direction incrementation variable for node <strong>N4</strong> on <strong>T-928a</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IXU</td>
<td>x-direction incrementation variable for use with the <strong>ID</strong> parameter on <strong>T-928a</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of the first four of these incrementation variables can be negative, zero or positive. **IXU** must be nonzero.

*go to T-928c*
**T-928c E928 UEL floatVariables**

This is where the user must specify the numeric value(s) of each of the nine floating point input variables that the developer of this type-928 UEL “created” for it (by defining the name and size of each of those variables in the *floatVariables* part of his *userElement* directive for that UEL (see the T-928 record description, above).

A single T-928c record must follow T-928a (or T-928b) for the current set of type-928 UELs.

<table>
<thead>
<tr>
<th>Area</th>
<th>cross section area (uniform along the beam)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iy</td>
<td>area moment of inertia about the element’s y axis</td>
</tr>
<tr>
<td>Iz</td>
<td>area moment of inertia about the element’s z axis</td>
</tr>
<tr>
<td>J</td>
<td>polar moment of inertia, about the element’s x axis</td>
</tr>
<tr>
<td>Material</td>
<td>material property-set identifier</td>
</tr>
<tr>
<td>ShearFactorY</td>
<td>shear-correction factor for the beam’s y-direction</td>
</tr>
<tr>
<td>ShearFactorZ</td>
<td>shear-correction factor for the beam’s z-direction</td>
</tr>
<tr>
<td>ECC(1)</td>
<td>eccentricity in the y-direction for each of the active nodes</td>
</tr>
<tr>
<td>ECC(2)</td>
<td>eccentricity in the z-direction for each of the active nodes</td>
</tr>
<tr>
<td>SCC(1)</td>
<td>shear center offset in the y-direction along the uniform cross section</td>
</tr>
<tr>
<td>SCC(2)</td>
<td>shear center offset in the z-direction along the uniform cross section</td>
</tr>
</tbody>
</table>

```plaintext
if  ( the user wants to add another set of E928 elements ) then
go to T-928
elseif ( the user wants to add a set of E940 elements ) then
go to T-940
elseif ( the user wants to add a set of E900 elements ) then
go to T-900
else
    return to T-100
endif
```
T-940 E940 MIN4 Quadrilateral UEL

STAGS’ E940 4-node quadrilateral element—shown in the following Figure—was originally implemented in the program as a User-Defined Element, in accord with the UEL specifications and conventions that are described here and in Chapter 13 of this document. After testing this User-Defined element successfully, it was incorporated into STAGS as a “built-in” UEL that anyone can use without having to re-make s1, s2 and other STAGS processors to do so.

This T-940 element-specification record (set) should be used instead of the T-900 record (set) described earlier in this section when the user wants to add one or more “built-in” type-940 UELs to his current element unit. STAGS uses the same subroutines—and the same data spaces—to read and process all of the “generic” and type-specific input records that are needed for any type of UEL, so what we are calling the T-940 record here is actually a T-900 record that should be used as described here.

Before we turn our attentions to this record (and the other records in the T-940 record set), we need to remind the user whose model has one or more E940 UELs in it that he must include the following *userElement and *userProperty UEL-definition directives somewhere in his case.inp file for that model prior to his first use of any E940 element:
These directives replicate the “standard” properties that the developer of this UEL specified when he implemented his 4-node MIN4 quad element in STAGS as a type-940 element. These element-definition directives are printed in Blue here to remind the user that they must be included in his case.inp file “precisely” as they are shown here (with no substantive changes) somewhere in that file before the first E840 element in the model is specified. The user does not have to include *userProperty directives in his case.inp file to specify geometric and/or material properties for those E840 elements because STAGS gets all of the material and fabrication information that it needs for those elements from the program’s comprehensive GCP facilities.

That done, the analyst can then use a single T-940 record (which is in fact a UEL-type-specific T-900 record) to specify that he wants to add one or more E940 elements to the current element unit—and to specify the “element number” for the first element in this set, the four input and control flags, and the two incrementation variables that he can use with E940 UELs:
<table>
<thead>
<tr>
<th>KELT</th>
<th>ID</th>
<th>IFAB</th>
<th>IANG</th>
<th>ILIN</th>
<th>IPLAS</th>
<th>NX</th>
<th>NY</th>
</tr>
</thead>
<tbody>
<tr>
<td>KELT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**KELT** must = 940 for this **E940 UEL** ................. see **Note 1**, below

**ID** starting element number for first element in this element set

**IFAB** fabrication identifier: must be < 0 for this **E940 UEL**, to identify a fabrication in the GCP Fabrication Table (I-22a)

**IANG** wall-reference option:

- 0 – use default strategy of projecting \( x_g, y_g \) to establish \( x_w \)
- 1 – specify an x-direction vector for use in establishing the material orientation matrix
- 2 – specify x- and y-direction vectors for use in establishing the material orientation matrix

**ILIN** geometric-nonlinearity flag:

- 0 – use nonlinear strain-displacement relations
- 1 – use linear strain-displacement relations

**IPLAS** material-nonlinearity flag:

- 0 – elastic behavior only
- > 0 – plasticity included

**NX** # of **E940s** to be generated in the x direction (default=1) ... see **Note 2**, below

**NY** # of **E940s** to be generated in the y direction (default=1) ... see **Note 2**, below

**Note 1:** Any **KELT** value other than 940 on this record tells **STAGS** that the user wants to stop adding **E940 elements** and start adding one or more type-**KELT UELs** to the current element unit. In this case, the user should follow the prescriptions given elsewhere in this section for that type of **UEL** (instead of following the prescriptions given here for type-940 elements).

**Note 2:** The user can generate a 2-dimensional “patch” of **E940 UELs** using the **NX and NY** incrementation variables on this record, but he cannot (currently) generate a “layered” set of 2-dimensional patches by using all three of the incrementation variables that the T-900 record offers; the prudent user should refrain from using T-900’s **NZ** variable—or set it equal to 1 if he cannot bear to do that.

---

*go to T-940a*
T-940a E940 UEL Nodes

This is where the user specifies the four node points that define the first E940 element that he wants to add to the current element unit via the current T-940 record set. STAGS “knows” that he must specify four nodes here because the *userElement directive that defines the attributes of a type-940 element specifies that each type-940 element in any given element unit must have four nodes—each of which is included in the nodal point list that he constructed via his S-1 or S-3 specifications for that unit.

Each T-940 record must be followed by a single T-940a E940 UEL NODES record:

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>node # 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>node # 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>node # 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N4</td>
<td>node # 4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NX (T-940)  x-direction incrementation flag
NY (T-940)  y-direction incrementation flag
IANG (T-940) wall-reference option

if ( NX > 1 ) then
go to T-940b
elseif ( NY > 1 ) then
go to T-940c
elseif ( IANG = 1 ) then
go to T-940d
elseif ( IANG = 2 ) then
go to T-940e
else
   follow the instructions at the end of T-940g
endif
**T-940b E940 UEL X-Direction Incrementations**

Incrementation variables are specified here for use with the \textbf{NX} (x-direction) looping function invoked on the T-900 record and the initial node points established on T-940a.

<table>
<thead>
<tr>
<th>IX1</th>
<th>IX2</th>
<th>IX3</th>
<th>IX4</th>
<th>IXU</th>
</tr>
</thead>
<tbody>
<tr>
<td>IX1</td>
<td>x-direction incrementation variable for node \textbf{N1} on T-940a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX2</td>
<td>x-direction incrementation variable for node \textbf{N2} on T-940a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX3</td>
<td>x-direction incrementation variable for node \textbf{N3} on T-940a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IX4</td>
<td>x-direction incrementation variable for node \textbf{N4} on T-940a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IXU</td>
<td>x-direction incrementation variable for use with the \textbf{ID} parameter on T-940a</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of the first four of these incrementation variables can be negative, zero or positive. \textbf{IXU} must be nonzero.

\begin{verbatim}
if ( \textbf{NY} > 1 ) then
go to T-940c
elseif ( \textbf{IANG} = 1 ) then
go to T-940d
elseif ( \textbf{IANG} = 2 ) then
go to T-940e
else
go to T-940f
endif
\end{verbatim}
**T-940c E940 UEL Y-Direction Incrementations**

Incrementation variables are specified here for use with the NY (y-direction) looping function invoked on the T-900 record and the initial node points established on T-940a.

<table>
<thead>
<tr>
<th>IY1</th>
<th>IY2</th>
<th>IY3</th>
<th>IY4</th>
<th>IYU</th>
</tr>
</thead>
<tbody>
<tr>
<td>IY1</td>
<td>y-direction incrementation variable for node N1 on T-940a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IY2</td>
<td>y-direction incrementation variable for node N2 on T-940a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IY3</td>
<td>y-direction incrementation variable for node N3 on T-940a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IY4</td>
<td>y-direction incrementation variable for node N4 on T-940a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IYU</td>
<td>y-direction incrementation variable for use with the ID parameter on T-940a</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of the first four of these incrementation variables can be negative, zero or positive. IYU must be nonzero.

**IANG (T-940) wall-reference option**

if ( IANG = 1 ) then
   go to T-940d

elseif ( IANG = 2 ) then
   go to T-940e

else
   go to T-940f
endif
**T-940d E940 UEL Material Orientation Vector**

If \( \text{IANG} = 1 \) (T-940), a single T-940d record must follow T-940a (or T-940b or T-940c) for the current set of E940 User elements. T-940d specifies a single x-direction vector that is used to establish the material orientation triad for each of the User elements that are to be generated by the current T-940 element-definition set.

\[
\begin{align*}
X_F & \quad X_F & \quad X_F \\
X_F & \quad X_F & \quad X_F
\end{align*}
\]

\( X_F, X_F, X_F \) vector components establishing the x orientation of the material

goto T-940f
**T-940e E940 UEL Material Orientation Vectors**

If \textit{iang} = 2 (T-940), a single T-940e record must follow T-940a (or T-940b or T-940c) for the current set of E940 User elements. T-940e specifies an x-direction and a y-direction vector that are \textit{both} used to establish the material orientation triad for each of the User elements that are to be generated by the current T-940 element-definition set.

\[
\begin{align*}
XFX & \quad XFY & \quad XFZ \\
YFX & \quad YFY & \quad YFZ
\end{align*}
\]

\textbf{XFX, XFY, XFZ} \quad vector components establishing the x orientation of the material

\textbf{YFX, YFY, YFZ} \quad vector components establishing the y orientation of the material

\textit{go to T-940f}
T-940f E940 UEL floatVariable

This is where the user must specify the numeric values of the two floating point input variables that the developer of this type-940 UEL “created” for it (by defining their names and sizes in the *floatVariables part of his *userElement directive for that UEL (see the T-940 record description, above).

A single T-940f record must follow T-940a (or T-940b or T-940c or T-940d or T-940e) for the current set of type-940 UELs.

---

**UniformPressure(1) UniformPressure(2)**

<table>
<thead>
<tr>
<th>UniformPressure(1)</th>
<th>P1 component of uniform pressure loading (positive in the +z direction in the element’s local coordinate system); see T-940g, on the following page</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniformPressure(2)</td>
<td>P2 component of uniform pressure loading (positive in the +z direction in the element’s local coordinate system); see T-940g, on the following page</td>
</tr>
</tbody>
</table>

---

**go to** T-940g


**T-940g E940 UEL integerVariables**

This is where the user must specify the numeric values of the two integer input variables that the developer of this type-940 UEL “created” for it (by defining their names and sizes in the *integerVariables* part of his *userElement* directive for that UEL (see the T-940 record description, above).

A single T-940g record must follow the T-940f record for the current set of type-940 UELs.

<table>
<thead>
<tr>
<th>IntegOrder</th>
<th>LoadType</th>
</tr>
</thead>
<tbody>
<tr>
<td>integration order flag:</td>
<td></td>
</tr>
<tr>
<td>= 0 – stabilized reduced integration strategy that works very well in practice (strongly recommended)</td>
<td></td>
</tr>
<tr>
<td>= 1 – single (centroidal) integration point (never recommended)</td>
<td></td>
</tr>
<tr>
<td>= 4 – full integration</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LoadType</th>
</tr>
</thead>
<tbody>
<tr>
<td>load type flag:</td>
</tr>
<tr>
<td>= -1 – ignore the P1 and P2 pressure loadings specified on the T-940f record; apply pressure loading(s) via the UPRESS user-written subroutine</td>
</tr>
<tr>
<td>= 0 – apply the P1 pressure loading via the P_a loading system</td>
</tr>
<tr>
<td>= 1 – apply the P1 pressure loading via the P_a loading system</td>
</tr>
<tr>
<td>= 2 – apply the P1 pressure loading via the P_a loading system and apply the P2 pressure loading via the P_b loading system</td>
</tr>
</tbody>
</table>

**Note 1:** The P1 and P2 uniform pressure loading components are specified via the UniformPressure(1) and UniformPressure(2) variables on the T-940f record (which is described on the preceding page)

if ( the user wants to add another set of E940 elements ) then
  go to T-940
elseif ( the user wants to add a set of E928 elements ) then
  go to T-928
elseif ( the user wants to add a set of E900 elements ) then
  go to T-900
else
  return to T-100
endif
This chapter contains descriptions of STAGS input requirements for specification of loadings that can be applied to an element unit, of specifications that can be made to control output produced by the program in generating and in processing an element unit, and of specifications that may be made to include externally-generated linear stiffness matrices in the model (after all of the element units in the model have been defined).

10.1 Element unit loadings

Element-unit loadings are in many ways similar to shell-unit loadings, which are described in Section 6.5 “Loads”. The following discussion of element-unit loadings contains frequent references to Section 6.5 in lieu of repeating documentation presented there. Differences between element-unit and shell-unit loadings are pointed out, where they occur—as are features which are unique to element units. For the most part, however, the user may simply refer to the indicated shell-unit loading documentation. Other information which will be found particularly useful in a study of element-unit loading techniques is summarized below:

- Coordinate systems—Section 4.1 “Coordinate Systems” on page 4-1, and Section 14.2 “Algorithm for Determining the Element Frame” on page 14-2.
- Element node-numbering and edge-numbering conventions—Chapter 14 “The Element Library”.
- Distributed loading—Section 4.6 “Loads” on page 4-11.
- Shell-units loads—Section 6.5 “Loads” on page 6-64.

The only type of surface traction that is permissible on shell elements is normal pressure, which may be input either as dead pressure (LT=4, U-3), which remains normal to the undeformed
surface, or as live pressure (LT=5), which remains normal to the deformed surface throughout large deformations. The user must be aware that the \((z')\) element normal determines the direction of pressure. That normal, in turn, is determined by the element node numbering (T-3, T-4). A complete discussion of node numbering can be found with those records. For line loads (LT=2), the situation is more complicated. The user is asked to provide the load direction (LD, U-3), the element number, and the element edge number. The user may select from three coordinate systems for expressing load directions—\((x'', y'', z'')\) computational coordinates, \((x_g, y_g, z_g)\) nodal-global coordinates, and \((x_e, y_e, z_e)\) element-edge coordinates. Choice of coordinate system is independently specified for each load record (LAX, U-3). As explained in Section 4.1 “Coordinate Systems”, the user may optionally (IAUX, S-1) define a \((x_{a}, y_{a}, z_{a})\) nodal-auxiliary coordinate system at each node. Where a nodal-auxiliary system is defined, it determines the computational system at that node. Otherwise, the nodal-global system is used. It is the user’s responsibility to define sensible auxiliary systems in order for the loading to be physically meaningful.

Prestress for a linear eigenvalue analysis (bifurcation buckling or small vibrations) may be defined for element units by two methods. As for shell units, a uniform stress state is defined on a U-5 record (\(\text{cf. Q-5, shell unit}\)). The prestress state is defined in \((\bar{x}, \bar{y})\) fabrication coordinates and is applied to all shell elements in the unit. A non-uniform prestress field can be defined using U-3 records, LT=3, again in fabrication coordinates. Both options are allowed within each element unit. It is important to note that if both U-5 and U-3 prestress are specified, the U-3 records override the constant field for those elements where they have been defined.

For the element unit, it is possible to specify incremental boundary conditions for bifurcation buckling that are different from those applied to the prebuckling state. See Section 6.4 “Boundary Conditions” on page 6-57. The discussion there pertaining to basic/incremental BCs in shell units is also applicable to element units, with \(dof\) constraints imposed \(\text{via specified displacements on U-3 (LT = -1)}\). As for shell units, the value of the displacement (\(P, U-3\)) is ignored, and a \(dof\) constraint (i.e., specified zero) is applied to the incremental BCs.
U-1 Loads Summary

This record summarizes the load data to be defined for the element unit. It is always included in each of the element units. Element-unit loading is similar to shell-unit loading. For information about that, see Section 10.1 “Element unit loadings” on page 10-1, and “Q-1 Loads Summary” on page 6-67.

| NSYS | number of load systems |
| NICS | number of sets of initial conditions |
| NAMS | number of attached masses |
| NUSS | NUSS > 0 indicates that a uniform basic stress state will be defined on U-5 for vibration or bifurcation buckling |
| NHINGE | number of cable hinge restraint vectors |
| NMOMNT | number of cable hinge moment loading vectors (must be 0, currently) |
| NLEAST | number of least-squares loading sets |
| IPRESS | 0 – no UPRESS 1 – UPRESS is included |

User-written subroutine UPRESS defines pressure–surface loading which acts normal to the element surface. The inclusion of UPRESS precludes the definition of pressure loading in the INP file. When IPRESS = 1, any pressure loads defined on Q-3/U-3 records (LT = 4 or 5) are ignored.

```
if ((NSYS > 0) or (NICS > 0)) then  go to  U-2
elseif (NAMS > 0) then  go to  U-4
elseif (NUSS > 0) then  go to  U-5
elseif (NHINGE > 0) then  go to  U-6
elseif (NMOMNT > 0) then  go to  U-7
elseif (NLEAST > 0) then  go to  U-8a
else  go to  V-1
```
U-2 Load Set Summary

Element-unit loading is similar to shell-unit loading. For information about that, see Section 10.1 “Element unit loadings” on page 10-1, and “Q-2 Load Set Summary” on page 6-69.

The number of loadsets, \( S_i \), is defined as \( S_i = \text{NSYS} + \text{NICS} \) (U-1), where \( \text{NSYS} \leq 2, \text{NICS} \leq 2 \); therefore \( S_i \leq 4 \). Each of the \( S_i \) load sets is defined in a U-2/ U-3 series. The order in which they are defined is established as:

- load system A
- load system B
- initial displacement
- initial velocity

All four load sets are optional. Each load set which is present must be completely defined via a U-2/3 series before moving on to the next set in the order specified above.

ISYS NN IFLG

ISYS for load systems:

1 – load system A
2 – load system B

for initial conditions:

0 – initial displacements
-1 – initial velocities

NN number of U-3 records needed to describe the loads or initial conditions

IFLG 0 – no USRLD
1 – loads defined in user-written subroutine USRLD are included

if (NN > 0) then go to U-3
else follow instructions at end of U-3
## U-3 Load Definition

This record is included only if **NN > 0** (U-2); it is repeated **NN** times. Element-unit loading is similar to shell-unit loading. For information about that, see Section 10.1 “Element unit loadings” on page 10-1, and “Q-3 Load Definition” on page 6-70.

Distributed loads are uniform over an element or over an element boundary. The location of a line load is given by the element edge number and the unit element number. Location of a surface traction is given by unit element number. Elements are automatically numbered by unit in the order that they are defined on the T records. The user can verify unit element numbering by referring to the **OUT1** (model output) file.

### P LT LD LI LJ LAX NDEFS INC1 INC2 INC3

<table>
<thead>
<tr>
<th>P</th>
<th>magnitude of load, displacement, or velocity; the load type and direction are determined by LT and LD, respectively.</th>
</tr>
</thead>
<tbody>
<tr>
<td>LT</td>
<td>-1  – prescribed displacement (translation or rotation), or initial condition (initial displacement or initial velocity)</td>
</tr>
<tr>
<td></td>
<td>1  – point force/moment</td>
</tr>
<tr>
<td></td>
<td>2  – line load/moment</td>
</tr>
<tr>
<td></td>
<td>3  – initial prestress</td>
</tr>
<tr>
<td></td>
<td>4  – dead pressure—normal to the undeformed element surface</td>
</tr>
<tr>
<td></td>
<td>5  – live pressure—normal to the deformed element surface throughout geometrically-nonlinear deformations</td>
</tr>
<tr>
<td></td>
<td>6  – velocity-dependent forces (see E-1) — INACTIVE</td>
</tr>
<tr>
<td></td>
<td>7  – surface traction — INACTIVE</td>
</tr>
</tbody>
</table>

> Note that when **LT = 3**, any uniform prestress defined on U-5 is overridden for the corresponding element.

<table>
<thead>
<tr>
<th>LD</th>
<th>load direction; interpretation of ((X, Y, Z)) is dependent upon LAX, below. <strong>LD</strong> must be set to 3 when <strong>LT \geq 4</strong>.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1  – (u); force/translation in the X direction. (N_{xx}) prestress if <strong>LT = 3</strong></td>
</tr>
<tr>
<td></td>
<td>2  – (v); force/translation in the Y direction. (N_{yy}) prestress if <strong>LT = 3</strong></td>
</tr>
<tr>
<td></td>
<td>3  – (w); force/translation in the Z direction. (N_{zz}) prestress if <strong>LT = 3</strong></td>
</tr>
<tr>
<td></td>
<td>4  – (ru); moment/rotation, right-handed about the X axis</td>
</tr>
<tr>
<td></td>
<td>5  – (rv); moment/rotation, right-handed about the Y axis</td>
</tr>
<tr>
<td></td>
<td>6  – (rw); moment/rotation, right-handed about the Z axis</td>
</tr>
</tbody>
</table>
LI  load location identifier

LT \leq 1: \text{ node number} \\
LT = 2: \text{ element edge number} \\
LT \geq 3: \text{ not used}

LJ  element number; irrelevant when LT \leq 1, LJ = 0 applies pressure to all elements for LT \geq 4.

LAX  load axes, determining the interpretation of LD, above

0 \quad (X, Y, Z) \text{ correspond to } (x'', y'', z'') \text{ computational coordinates} \\
1 \quad (X, Y, Z) \text{ correspond to } (x_g, y_g, z_g) \text{ nodal-global coordinates} \\
2 \quad (X, Y, Z) \text{ correspond to } (x_e, y_e, z_e) \text{ element-edge coordinates}

LAX = 0 \text{ is required for } LT = -1 \\
LAX = 2 \text{ is permitted for line loads only}

For LT = 3, the prestress state is defined in \((\bar{x}, \bar{y})\) fabrication coordinates. The \((z')\) element normal defines the direction of pressure.

NDEFS  number of loading definitions specified \textit{via} the current U-3 record; set equal to unity by STAGS if less than 2 or if omitted

INC1  incrementation parameter for use with the LD variable

INC2  incrementation parameter for use with the LI variable

INC3  incrementation parameter for use with the LJ variable

\begin{align*}
\text{NAMS} & (U-1) \quad \text{number of attached masses} \\
\text{NUSS} & (U-1) \quad \text{uniform basic stress state option for eigenanalysis} \\
\text{NHINGE} & (U-1) \quad \text{number of cable hinge restraint vectors} \\
\text{NMOMNT} & (U-1) \quad \text{number of cable hinge moment loading vectors} \\
\text{NLEAST} & (U-1) \quad \text{number of least-squares loading sets} \\
\end{align*}

\begin{align*}
\text{if} \quad (\text{NAMS} > 0) & \text{ then } \quad \text{go to } U-4 \\
\text{elseif} \quad (\text{NUSS} > 0) & \text{ then } \quad \text{go to } U-5 \\
\text{elseif} \quad (\text{NHINGE} > 0) & \text{ then } \quad \text{go to } U-6 \\
\text{elseif} \quad (\text{NMOMNT} > 0) & \text{ then } \quad \text{go to } U-7 \\
\text{elseif} \quad (\text{NLEAST} > 0) & \text{ then } \quad \text{go to } U-8a \\
\text{else} & \quad \text{go to } V-1 \\
\end{align*}
**U-4 Attached Mass**

The U-4 records allow the user to define attached masses at any node point in the element unit. The record is repeated NAMS (U-1) times.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GM</td>
<td>NM</td>
<td>NDEFS</td>
<td>INC</td>
</tr>
<tr>
<td>weight of attached mass, in units of <em>force</em></td>
<td>node number of attached mass</td>
<td>number of masses to be attached</td>
<td>incrementation parameter for the NM variable</td>
</tr>
</tbody>
</table>

NUSS (U-1) uniform basic stress state option for eigenanalysis
NHINGE (U-1) number of cable hinge restraint vectors
NMOMNT (U-1) number of cable hinge moment loading vectors
NLEAST (U-1) number of least-squares loading sets

if (NAMS attached-mass records have been defined) then
if (NUSS > 0) then go to U-5
elseif (NHINGE > 0) then go to U-6
elseif (NMOMNT > 0) then go to U-7
elseif (NLEAST > 0) then go to U-8a
else go to V-1
else continue defining U-4
**U-5 Uniform Stress State for Eigenanalysis**

This record may be used to define the stress resultants in a uniform basic stress state for bifurcation buckling or small vibration analysis. When a U-5 record is included, the only type of loading allowed is specified displacements \((LT = -1, \ U-3)\). Any other loading (mechanical or thermal) which may be defined is ignored, not only in the current unit, but in all units. The stress resultants defined on this record are multiplied by the load factors \(STLD(1), STLD(2)\) \((C-1)\) for load systems A and B, respectively.

The prestress state is defined in \((\tau, \gamma)\) fabrication coordinates.

<table>
<thead>
<tr>
<th>PNXA</th>
<th>PNYA</th>
<th>PNXYA</th>
<th>PNXB</th>
<th>PNYB</th>
<th>PNXYB</th>
</tr>
</thead>
<tbody>
<tr>
<td>value of the stress resultant in the (\tau) direction, load system A</td>
<td>value of the stress resultant in the (\gamma) direction, load system A</td>
<td>value of the shear resultant, load system A</td>
<td>value of the stress resultant in the (\tau) direction, load system B</td>
<td>value of the stress resultant in the (\gamma) direction, load system B</td>
<td>value of the shear resultant, load system B</td>
</tr>
</tbody>
</table>

\[\text{NHINGE \hspace{1cm} (U-1) \hspace{1cm} number of cable hinge restraint vectors}\]
\[\text{NMOMNT \hspace{1cm} (U-1) \hspace{1cm} number of cable hinge moment loading vectors}\]
\[\text{NLEAST \hspace{1cm} (U-1) \hspace{1cm} number of least-squares loading sets}\]

\[
\text{if \hspace{1cm} (NHINGE > 0) \hspace{1cm} then \hspace{2cm} go to \hspace{1cm} U-6} \\
\text{elseif \hspace{1cm} (NMOMNT > 0) \hspace{1cm} then \hspace{2cm} go to \hspace{1cm} U-7} \\
\text{elseif \hspace{1cm} (NLEAST > 0) \hspace{1cm} then \hspace{2cm} go to \hspace{1cm} U-8a} \\
\text{else \hspace{2cm} go to \hspace{1cm} V-1}
\]
U-6 Cable Hinge Restraint

This record is included only if \( \text{NHINGE} > 0 \) (U-1); it is repeated \( \text{NHINGE} \) times.

See discussion of cable hinge restraints under U-1. The vector described below gives the direction of the constraint in \((x'', y'', z'')\) computational coordinates. Although the magnitude of this vector does not theoretically influence the result, this is a multipoint constraint imposed by Lagrange multipliers. The new equation (automatically) produced by this constraint is assembled into the stiffness matrix along with the other freedoms. To avoid numerical problems, the magnitude of this vector is converted to a scaling constant to help the conditioning of the stiffness matrix. See the general discussion of Lagrange constraints under G-3 for more details.

<table>
<thead>
<tr>
<th>IHND</th>
<th>HRU</th>
<th>HRV</th>
<th>HRW</th>
<th>NDEFS</th>
<th>INC</th>
</tr>
</thead>
<tbody>
<tr>
<td>IHND</td>
<td>node number where constraint is applied</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HRU, HRV, HRW</td>
<td>the direction vector, in ((x'', y'', z'')) computational coordinates, along which rotation is prevented; the magnitude of this vector should be of the same order as the element torsional stiffness</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDEFS</td>
<td>number of constraints to be applied</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IHND</td>
<td>incrementation parameter for the IHND variable</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{NHINGE} \] (U-1) number of cable hinge restraint vectors
\[ \text{NMOMNT} \] (U-1) number of cable hinge moment loading vectors
\[ \text{NLEAST} \] (U-1) number of least-squares loading sets

if \( \text{NHINGE} \) (cable hinge restraint vectors have been defined) then
   if \( \text{NMOMNT} > 0 \) then \( \text{go to} \) U-7
   elseif \( \text{NLEAST} > 0 \) then \( \text{go to} \) U-8a
   else \( \text{go to} \) V-1
else \( \text{continue defining} \) U-6
**U-7 Cable Hinge Moment**

*NOTE*: This feature is currently disabled, and the user must set \( \text{NMOMNT} = 0 \) (U-1). *STAGS* automatically introduces cable moments for all applied moment loadings.

This record is included only if \( \text{NMOMNT} > 0 \) (U-1); it is repeated \( \text{NMOMNT} \) times.

See discussion of cable hinge moment loads under U-1. The vector described below gives the direction and magnitude of the applied moment in \((x'', y'', z'')\) computational coordinates.

<table>
<thead>
<tr>
<th>IMND MSYS RUM RVM RWM NDEFS INC</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMND node number where moment is applied</td>
</tr>
<tr>
<td>MSYS (1) – load applies to the A system (2) – load applies to the B system</td>
</tr>
<tr>
<td>RUM, RVM, RWM moment vector components in ((x'', y'', z'')) computational coordinates</td>
</tr>
<tr>
<td>NDEFS number of moments to be applied</td>
</tr>
<tr>
<td>IMND incrementation parameter for the IMND variable</td>
</tr>
</tbody>
</table>

\(\text{NMOMNT}\) (U-1) number of cable hinge moment loading vectors

\(\text{NLEAST}\) (U-1) number of least-squares loading sets

if \(\text{NMOMNT}\) cable hinge moment loading vectors have been defined) then

if \(\text{NLEAST} > 0\) then \text{go to U-8a}

else \text{go to V-1}

else \text{continue defining U-7}
U-8a Least Squares Loading Summary

There are NLEAST sets of U-8 records. Record U-8a is used to define the number NSQR of U-8b special loading records to follow, to define the reference node for the application of the least-squares summary loads or reactions, and to provide an overall scale factor SCALE for the six Lagrange constraints introduced automatically for each least-squares load set. The reference node must have been defined in the current or a previous unit. If any nodes not yet defined are referenced in any U-8 records, an error will result; to avoid this, postpone the definition of this least-squares set until after all pertinent nodes have been defined. As with all other Lagrange constraints that contribute to the stiffness matrix (see, for example, record G–3 and the ones that follow it), numerical roundoff considerations require that the values of the Lagrange unknowns not differ from other unknowns by too many orders of magnitude. This in turn means that the stiffness contributions should be near the same order as other members, or a SCALE value set to about equal to some average thickness times a modulus. The user should remember that this is only an order-of-magnitude estimate for numerical conditioning.

NSQR  IUNIT  IROW  ICOL  SCALE

NSQR number of U-8b data records required to define the edges involved in this least-squares load set.
IUNIT unit number for the reference node.
IROW row number for the reference node; if the reference node is in an element unit, IROW is the user point number (IUPT, S–1).
ICOL column number for the reference node; zero if node is in an element unit.
SCALE scale factor for numerical conditioning; we suggest something of the order of magnitude of the material modulus times a representative shell thickness.

go to  U-8b
U-8b Least Squares Load Definition

A least-squares constraint loading condition is specified by defining the following three components:

- A reference node with section weighting (see U-8a)
- A curve consisting of one or more arc segments and/or single nodes (U-8b).
- Loads on the reference node (see U-1, U-2, U-3 records)

U-8b input records define a curve consisting of one or more arc segments, or alternatively, individual nodes. There are a total of $\text{NSQRS}$ U-8b records. A sequence of U-8b records, all with $(\text{LU}, \text{LR}, \text{LC})$ defining actual nodes, specifies one arc segment. Additional arc segments may then be defined by adding a U-8b record with $(\text{LU}, \text{LR}, \text{LC}) = 0$ followed by more U-8b records defining the next arc segment. If a segment consists of a single node, the node weight $P$ is used unchanged. An individual U-8b record can specify several nodes at once in two ways:

- For shell units, $\text{LR}=0$ and $\text{LC}>0$ means the entire column $\text{LC}$, while $\text{LR}>0$ and $\text{LC}=0$ means the entire row $\text{LR}$.
- For either shell or element units, items $\text{LNDA}$, $\text{LNDB}$, and $\text{LNDINC}$ may be used to describe a range of nodes.

<table>
<thead>
<tr>
<th>P</th>
<th>LU</th>
<th>LR</th>
<th>LC</th>
<th>LNDA</th>
<th>LNDB</th>
<th>LNDINC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>Nodal weight. For sections of uniform composition, input 1.0. When the defining curve belongs to several distinct wall types, we suggest using 1.0 for the nodes in the stiffest structural areas and choosing other weights so that the set of P values are proportional to the modulus times the effective thickness ($E_h$).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LU</td>
<td>Unit number of node (shell unit; or element unit if $\text{LU} &gt; \text{NUNITS}$, B–2). If $\text{LU} = 0$, terminate the arc segment. Any U–8b records that follow describe a new arc segment to be added to this load set.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LR</td>
<td>Row number of first node (not used in element unit)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LC</td>
<td>Column number of first node (not used in element unit)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LNDA</td>
<td>First node of (row if $\text{LR} = 0$) or (column if $\text{LC} = 0$) in shell unit. First node in element unit ($\text{LR} = 0$). If 0 for a shell unit, the first node on the row or column is chosen. Must be nonzero for an element unit.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**LNDB**
Last node of (row if $LR=0$) or (column if $LC=0$) in shell unit. Last node in element unit ($LR=0$). If 0 for a shell unit, the last node on the row or column is chosen. Must be nonzero for an element unit. If $LNDB=LNDA>0$, then only one node is specified by this record.

**LNDINC**
Increment for nodes from $LNDA$ to $LNDB$. Increasing or decreasing ranges are allowed. If $LNDINC=0$, an increment of +1 is used for increasing ranges, and -1 for decreasing ranges.

**NSQR**
(U-8a) number of U-8b records

**NLEAST**
(U-1) number of least-squares loading sets

if (NSQR U-8b records have been defined) then
  if (NLEAST loading sets have been defined) then  go to V-1
  else  return to U-8a
else  continue defining U-8b
10.2 Output Control

The last information required for an element unit is the small set of type V records on which output-control parameters are specified for the unit. As with shell units, the values of these output-control parameters (and the number of V-type records) can vary from element unit to element unit.
V-1 Output Control—Record 1

This record is always included for each unit. The constants governing the output can differ from one unit to another.

Element-unit output control is similar to that for shell units. For more information about this, please refer to “R-1 Output Control—Record 1” on page 6-89.

<table>
<thead>
<tr>
<th>IPRD</th>
<th>IPRR</th>
<th>IPRE</th>
<th>IPRS</th>
<th>IPRP</th>
<th>NSELD</th>
<th>NSELS</th>
<th>IPRDSP</th>
<th>IPRSTR</th>
<th>ISL</th>
<th>ISS</th>
<th>ISD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>&gt;0</td>
<td>&gt;0</td>
<td>&gt;0</td>
<td>&gt;0</td>
<td>&gt;0</td>
<td>NSELD</td>
<td>NSELS</td>
<td>IPRDSP</td>
<td>IPRSTR</td>
<td>ISL</td>
<td>ISS</td>
<td>ISD</td>
</tr>
</tbody>
</table>

IPRD 0 – do not print displacements  
>0 – displacements are printed at every IPRD load or time step  

IPRR 0 – do not print stress resultants  
>0 – stress resultants are printed at every IPRR load or time step  
(elastic analysis only)  

IPRE 0 – do not print strains  
>0 – strains are printed at every IPRE load or time step  

IPRS 0 – do not print stresses  
>0 – stresses are printed at every IPRS load or time step  

Stress output is obtained only where indicated for the corresponding shell wall or beam cross-section. Please check LSOL (K-2), NSOYZ (J-1), ISP (J-3a), and ISOC (J-3b). Also, see ISL, ISS, and ISD, below.  

IPRP 0 – no additional stress output for points with yield  
>0 – stresses and strains are printed at all points with yield at every IPRP load or time step  

IPRF 0 – do not print nodal point forces (internal force vector)  
>0 – nodal point forces are printed at every IPRF load or time step  

NSELD number of records defining selected displacements (one record may correspond to a node, a row, or a column)  

NSELS number of records defining selected stresses
IPRDSP 0 – print selected displacements at every load or time step  
>0 – print selected displacements at every IPRDSP\textsuperscript{th} load or time step  
IPRSTR 0 – print selected stresses at every load or time step  
>0 – print selected stresses at every IPRSTR\textsuperscript{th} load or time step (disabled)  
ISL 0 – element results are computed at centroids  
1 – element results are computed at integration points  
ISS 0 – no transverse shear stresses  
1 – compute transverse shear stresses — INACTIVE  
ISD 0 – print stress and strain components in $(\bar{x}, \bar{y})$ fabrication coordinates  
1 – print stress components in both $(\bar{x}, \bar{y})$ fabrication coordinates and $(\phi_1, \phi_2)$ material coordinates  
2 – print stress components in both $(x, y)$ fabrication coordinates and the principal directions (includes angle of orientation)  

NUNITE (B-2) number of element units  
NSTIFS (B-2) number of linear-stiffness-matrix contributions  
if (NSELD > 0) then \textit{go to V-2}  
elseif (NSELS > 0) then \textit{go to V-3}  
else  
if (less than NUNITE element units have been defined) then \textit{return to S-1}  
elseif (NSTIFS > 0) then \textit{go to W-1}  
else \textit{data deck is complete}  
endif
### V-2 Output Control—Record 2

This record defines a number of nodes at which displacement are to be printed at each load or time step. It is repeated \textsc{NSELD} times (V-1).

<table>
<thead>
<tr>
<th>INOD1</th>
<th>INOD2</th>
<th>INODI</th>
</tr>
</thead>
<tbody>
<tr>
<td>INOD1</td>
<td>node number of first node in group</td>
<td></td>
</tr>
<tr>
<td>INOD2</td>
<td>node number of last node in group</td>
<td></td>
</tr>
<tr>
<td>INODI</td>
<td>node number increment</td>
<td></td>
</tr>
</tbody>
</table>

- \textsc{NSELD} (V-1) number of selected displacements
- \textsc{NSELS} (V-1) number of selected stresses
- \textsc{NUNITE} (B-2) number of element units
- \textsc{NSTIFS} (B-2) number of linear-stiffness-matrix contributions

if (less than \textsc{NSELD} V-2 records have been defined) then continue defining V-2
elseif \textsc{NSELS} > 0 then go to V-3
elseif (less than \textsc{NUNITE} shell units have been defined) then return to S-1
elseif \textsc{NSTIFS} > 0 then go to W-1
else data deck is complete
endif
**V-3 Output Control—Record 3**

This record defines a number of locations at which stresses will be printed at each load or time step. It is repeated \texttt{NSELS} times (V-1).

<table>
<thead>
<tr>
<th>IELS1</th>
<th>IELS2</th>
<th>IELSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>IELS1</td>
<td>element number of first element in group</td>
<td></td>
</tr>
<tr>
<td>IELS2</td>
<td>element number of last element in group</td>
<td></td>
</tr>
<tr>
<td>IELSI</td>
<td>element number increment</td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{NSELS} \quad (V-1) \quad \text{number of selected stresses}
\]
\[
\text{NUNIT} \quad (B-2) \quad \text{number of element units}
\]
\[
\text{NSTIFS} \quad (B-2) \quad \text{number of linear-stiffness-matrix contributions}
\]

- if ( less than \texttt{NSELS} \texttt{V-3} records have been defined ) then \textit{continue defining} \texttt{V-3}
- elseif ( less than \texttt{NUNIT} element units have been defined ) then \textit{return to} \texttt{S-1}
- elseif ( \texttt{NSTIFS} > 0 ) then \textit{go to} \texttt{W-1}
- else \textit{data deck is complete}
10.3 Linear-Stiffness Contributions

This information is required if and only if the NSTIFS parameter is positive on the user’s B-2 record. A positive value for NSTIFS tells STAGS that NSTIFS externally-generated “linear-stiffness contribution” definitions are required to complete the model—where each such contribution is a supermatrix of linear stiffness matrices coupling all of the freedoms at each of the nodes with which they are associated.

A single W-1 record is required for each linear-stiffness contribution definition.

This W-1 record must be followed by any required W-2a/W-2b or W-3 records, as described below.

All of the linear-stiffness-contribution definition records are read after the last (output control) record for the final element unit in the user’s model.
W-1 Linear-Stiffness Contribution—Record 1

This record is required for each of the NSTIFS (B-2) “linear-stiffness contributions” used for the entire model. A linear-stiffness contribution in this context is an externally-generated, \( \text{NRNOD} \times \text{NRNOD} \) supermatrix, each \( \text{NRDOF} \times \text{NRDOF} \) submatrix \( K_{ij} \) of which contains linear stiffness values for the \( i \)th and \( j \)th of the \( \text{NRNOD} \) nodes to which \( K \) contributes. Each of these nodes must have \( \text{NRDOF} \) degrees of freedom, where \( \text{NRDOF} \) is typically 3 or 6.

<table>
<thead>
<tr>
<th>NRDOF</th>
<th>NRNOD</th>
<th>NRKIJ</th>
<th>KLSTF</th>
<th>NRDIS</th>
<th>NRFOR</th>
</tr>
</thead>
</table>

NRDOF number of freedoms at each of the nodes to which \( K \) contributes; typically, \( \text{NRDOF} = 3 \) or 6; \( \text{NRDOF} \leq 6 \)

NRNOD number of nodes to which \( K \) contributes

NRKIJ connectivity control flag (see the note for the KLSTF parameter, below):

- \( \text{NRKIJ} \geq 0 \) read connectivity and stiffness data for this contribution
- \( \text{NRKIJ} < 0 \) read connectivity data only

KLSTF connectivity and (optionally) contribution control flag:

- \( \text{KLSTF} = 0 \) read nodal connectivities and stiffness-contribution values;
- \( 0 < \text{KLSTF} < N \) (where this is the \( N \)th of NSTIFS W-1 records) \( \Rightarrow \) use the values given for \( K \) matrix number \( \text{KLSTF} \) for the current case (where contribution values must have been specified explicitly for case \( \text{KLSTF} \); \( \text{KLSTF} = N \) read nodal connectivities and stiffness-contribution values

Note: STAGS sets \( \text{NRKIJ} = -1 \) if KLSTF is not equal to \( N \)

NRDIS identifier of reduced-displacement vector (not implemented yet)

NRFOR identifier of reduced-force vector (not implemented yet)

NSTIFS (B-2) number of linear-stiffness-matrix contributions

if ( NSTIFS W-1 records have been defined ) then
data deck is complete
else
  if ( \( \text{NRKIJ} > 0 \) ) then go to W-2a
  else
    go to W-3
endif
**W-2a Linear-Stiffness Contribution—Record 2a**

A W-2a record and its companion W-2b record are required to specify the nodal connectivity and stiffness values for each of the active submatrices $K_{ij}$ that comprise the $[K]$ supermatrix for the current linear stiffness contribution.

The two nodes $I = (IUNIT,IROW,ICOL)$ and $J = (JUNIT,JROW,JCOL)$ for each active submatrix in the upper or lower triangular part of $[K]$ are specified via the W-2a record, and the stiffness values are specified via W-2b. A submatrix is active if it has one or more nonzero stiffness values. Since the number of active submatrices is not known a priori, STAGS expects to read a user-controlled sequence of W-2a records—each followed by a companion W-2b record—with the user terminating the sequence via the IUNIT parameter, as described below.

```
IUNIT  IROW  ICOL    JUNIT  JROW  JCOL

IUNIT    unit number for node I; IUNIT ≤ 0 terminates the reading of W-2a input records
IROW     row number for node I, if IUNIT is a shell unit; or I node number, if IUNIT is an element unit
ICOL     column number for node I, if IUNIT is a shell unit; or ignored, if IUNIT is an element unit
JUNIT    unit number for node J, if JUNIT is a shell unit; or J node number, if JUNIT is an element unit
JROW     row number for node J, if JUNIT is a shell unit; or J node number, if JUNIT is an element unit
JCOL     column number for node J, if JUNIT is a shell unit; or ignored, if JUNIT is an element unit

NSTIFS (B-2) number of linear-stiffness-matrix contributions

if  ( IUNIT>0 )  then  go to W-2b
elseif ( NSTIFS linear-stiffness-contributions have been defined )then
   data deck is complete
else     go to W-1
endif
```
**W-2b Linear-Stiffness Contribution—Record 2b**

A W-2b record is required to specify the stiffness values for the active submatrix $K_{ij}$ that is identified by its above-described W-2a record.

\[
\begin{pmatrix}
(K_{IJ}(m,n), \text{n=1,NRDOF}), \text{m=1,NRDOF}
\end{pmatrix}
\]

**KIJ ... NRDOF × NRDOF** submatrix of linear-stiffness-contribution values

*go to W-1*
**W-3 Linear-Stiffness Contribution—Record 3**

A set of \( \text{NRNOD} \) (W-1) W-3 records is required to specify the nodal connectivities of a linear-stiffness contribution that uses the same stiffness values as a previously-specified contribution.

<table>
<thead>
<tr>
<th>IUNIT( (n) )</th>
<th>IROW( (n) )</th>
<th>ICOL( (n) )</th>
<th>( n = 1, 2, 3, \ldots, \text{NRNOD} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit number for node ( n )</td>
<td>row number for node ( n ), if IUNIT( (n) ) is a shell unit; or node number, if IUNIT( (n) ) is an element unit</td>
<td>column number for node ( n ), if IUNIT( (n) ) is a shell unit; or ignored, if IUNIT( (n) ) is an element unit</td>
<td></td>
</tr>
</tbody>
</table>

*go to W-1*
11 Solution Input

11.1 Solution Options

Linear solutions require only the computation and assembly of the stiffness matrix and the solution of a linear system of algebraic equations with the displacements \( \mathbf{d} \) as unknowns:

\[
K \mathbf{d} = \mathbf{f}_e
\]  

(11.1)

where \( \mathbf{f}_e \) is the external force or load vector.

STAGS has two independent load sets that are combined using load-factors input on the Load Multiplier record, C-1 described on page 11-11. For a linear solution, the total load on the system is \( \text{“load set A” } \times \text{STLD(1)} + \text{“load set B” } \times \text{STLD(2)} \). Except for thermal loads, no other information on the C-1 record is relevant. Setting \( \text{INDIC} = 0 \) on record B-1 causes a linear stress analysis to be carried out.

There are two types of eigenvalue analyses: bifurcation buckling and vibration. To give the user more flexibility, eigenvalue strategy records are provided, which include data that controls the error or iterative change in the eigenvalue tolerance, the range and number of eigenvalue-eigenvector pairs, how much diagnostic data to print, and how much computer time can be spent during the run. Bifurcation analyses require a stress state, which can be linear or nonlinear. A linear bifurcation analysis will be performed if \( \text{INDIC} = 1 \) on record B-1. In this case, STAGS will first perform a linear stress analysis using the loading case A. The linear stress state from that analysis is then used in generating the stability matrix required for the eigenanalysis. Alternatively, a uniform stress state can be specified for the shell elements and the stability matrix formed using that uniform state. Critical loading combinations are also output for each eigenvalue computed. For nonlinear bifurcation, the stress state from a previously-computed nonlinear static solution is used for the stiffness and stability matrices. The second type of eigenproblem treated by STAGS is the small vibration problem, which can be conducted either from a stress-free state (\( \text{INDIC} = 2 \)), or from a pre-stressed state (\( \text{INDIC} = 5 \)).
Nonlinear static solutions derive their nonlinearity from four sources. First, displacements may be so large that the force generated from deflections is no longer proportional to these deflections. This type of geometric nonlinearity arises from the kinematics of the problem, i.e., from the strain-displacement relations. The second source of nonlinearity stems from nonproportional material behaviors and can include hyperelasticity, plastic flow, creep, and progressive damage. The third source of nonlinearity comes from follower loading, including “live” pressures, where the direction of the loading depends on the displacement unknowns. The fourth source of nonlinearity stems from changing boundary conditions—otherwise known as contact and slip. Whatever the cause of nonlinearity, instead of a simple set of linear equations to solve, the program must solve the nonlinear equilibrium equations

\[ f_i(d) - f_e(d) = 0 \] (11.2)

where the vector \( f_i(d) \) is the internal force, with a component for each freedom in the system, and where dependence on the system displacements \( d \) is indicated. Since there is no direct way to solve such a system, the final solution must be obtained in steps. These equations are typically solved with an incremental-iterative procedure such as the Newton-Raphson method using the two basic techniques

- load incrementation
- iteration from a linearized equation system

Load parameter incrementation involves the choice of the independent state variable that controls the loading environment of the structure. In load control, both the A and B load systems are independently varied, using load steps whose magnitude depends on the success or failure to obtain a converged solution. For cases where the load reaches some maximum (limit) point along the solution path, load control breaks down and must be replaced by a state variable that more closely reflects the system response.

The preferred method for nonlinear static analysis is the so-called path-parameter strategy, where an increment of arc-length along the solution path replaces load as the independent variable. The A system load factor \( P_A \) is used as the new unknown. The B system is tied to the A system, and “floats” along with the A system until it reaches one of the set limits, after which it is frozen. If the A system is fixed, the B system load factor \( P_B \) is used as the independent path parameter. The constant of proportionality between the A and B systems is determined by the increments input on the C-1 record. Users need to understand that the phrase “load factor” is used in a generic sense. STAGS provides a capability to apply external loads as well as specified displacements. The solution control strategy (see NSTRAT on the D-1 record for both cases is referred to as “load control” even though in the latter case it is actually “displacement control.”

In all cases, nonlinear solutions are initiated from a linear solution, and load or arc-length increments are automatically adjusted as functions of the solution behavior.
Once a loading is chosen, a trial solution is selected by extrapolating the previous solutions based on the increments selected. Usually, this trial vector will not satisfy equilibrium equations, and a better approximation must then be computed. STAGS does this by linearizing the problem about the trial state and beginning with a modified- or true-Newton iteration sequence. If the trial solution is close enough, the sequence of approximate solutions will converge, as evidenced by a small value of the norm of both the displacement increment and the residual error.

For modified Newton, the stiffness matrix is computed only occasionally, often at the beginning of the solution step or when the solution fails to converge; the user has control of how often this is done (via record D-1).

For true Newton, a new stiffness is computed at each iteration. Even for modest systems, this is generally very expensive. There are instances, however, where because of sudden changes in system response, the only way out of difficulty is to take advantage of the enhanced quadratic convergence that the true-Newton iteration method yields. The most common strategy in these situations is to use the true-Newton method for a few steps, and then switch back to less intensive methods when the system settles down.

In any case, iteration stops either when the relative equilibrium and displacement norm ratios are within a specified error tolerance, \textbf{DELX} on record D-1, or after either divergence occurs or too many iterations are required. If no convergence is obtained, STAGS will compute a new tangent stiffness matrix based on the best available estimate of the solution. If that procedure fails, the code will try to solve the system using a smaller load or arc-length increment. After a specified number of repetitions (or cuts), also under user control, the code will give up and save results for subsequent restart.

Transient analyses involve the time integration of the initial value problem in time using a numerical time integrator. There are two types of time integrators:

- **explicit** \text{INACTIVE}
- **implicit**

Although explicit integrators have the advantage of not requiring a solution of an equation system (no stiffness matrix necessary), they suffer the usually severe numerical stability limit on the time increment, often making explicit integration much more expensive than the more complex implicit integration. Because explicit integrators track the response well throughout the frequency spectrum, they are very effective for early-time response. Currently, explicit time integration is not supported in STAGS.

Implicit integrators involve the solution of a nonlinear system that closely resembles its static analysis counterpart, requiring the complexity of a nonlinear solution iteration sequence. The advantage is that most implicit integrators are unconditionally stable, allowing much larger time
Solution-branch-switching algorithms offer the user the opportunity to jump from one solution path to another in the vicinity of a bifurcation point. Unlike simple limit points, bifurcation points are places where more than one solution exists for a given load state. If the desired response is on one of these alternate solution branches, one must be able to switch to it without convergence difficulties. One method is to introduce initial geometric imperfections. Most of the time, the imperfections will smooth out the solution, permitting a straightforward analysis along the minimum energy solution path. If branch switching is required, the Equivalence Transformation bifurcation processor (ET) can be used. ET avoids convergence problems by removing the variable that has the greatest magnitude in the local bifurcation mode and replacing it by an unknown scale factor times the eigenvector. In the majority of cases, the switch to the alternate path proceeds smoothly and efficiently.

Finally, STAGS provides the Load Relaxation option. If for some reason the system is left in a state far from equilibrium (as happens with the opening of a crack or with the switch from a transient to a static analysis), it is usually impossible to obtain a converged solution using the standard nonlinear solution techniques. Load relaxation allows the gradual transition from the non-equilibrium state to the desired equilibrium solution using the same solution methods that are available for ordinary static analysis.

### 11.2 Summary and Control Parameters

Two records in this group (A-1 and B-1) are always included, and a third record (B-2) is optional. The A-1 and B-1 records that appear in the BIN (solution input) file are similar to—but can and frequently do contain different information from—the A-1 and B-1 records for the INP (model input) file. The B-2 record, here, is very different from that in the INP file.

A-1, the Case Title record, contains a “case-title” character string that is included strictly for the user’s convenience. The B-1 (Analysis Type Definition) record contains parameters that define the type of analysis to be performed and (optionally) solution information to be saved and/or printed and (also optionally) whether or not the user exercises choice or control over the equation solver to be used.

The B-1 and B-2 records in the BIN file are *not* interchangeable with the B-1 and B-2 records in the INP file: they contain different input data altogether.
A-1 Case Title

The case title is read on the first line, which may contain any alphanumeric characters. This text is printed at the beginning of the output for the case and for identification on any disk file saved for possible restart. Subsequently any number of comment records can be added provided they begin with a “$” character in column 1. Comments can also be included at the end of a data line—a “$” character terminating data, and the comment following. A list of the complete input file, including any comment records, is printed at the beginning of the case.out2 text output file. The user is urged to use this record as a way to document the analysis.
**B-1 Analysis Type Definition and High-Level-Control Data**

The B-1 record for the BIN (solution input) file contains basic analysis-type- and high-level-control-definition information and is *not* interchangeable with the B-1 record for the INP (model input) file: they contain different input data altogether.

<table>
<thead>
<tr>
<th>INDIC</th>
<th>analysis-type definition parameter:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>linear analysis</td>
</tr>
<tr>
<td>1</td>
<td>bifurcation buckling analysis (linear stress state)</td>
</tr>
<tr>
<td>2</td>
<td>small vibrations (stress free state)</td>
</tr>
<tr>
<td>3</td>
<td>nonlinear static analysis</td>
</tr>
<tr>
<td>4</td>
<td>provide nonlinear solutions at specified load levels</td>
</tr>
<tr>
<td>5</td>
<td>small vibrations (linear or nonlinear stress state)</td>
</tr>
<tr>
<td>6</td>
<td>transient response analysis (geometrically nonlinear)</td>
</tr>
<tr>
<td>7</td>
<td>transient response analysis (geometrically linear)</td>
</tr>
<tr>
<td>8</td>
<td>special purpose (experimental)</td>
</tr>
<tr>
<td>9</td>
<td>special purpose (experimental)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IPOST</th>
<th>displacement data archival switch:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>do not save displacement data</td>
</tr>
<tr>
<td>0</td>
<td>save displacement data every IPOSTth step.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ILIST</th>
<th>data printout option:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>normal printout</td>
</tr>
<tr>
<td>1</td>
<td>full printout</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ICOR</th>
<th>corotation option:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>corotational procedure is used</td>
</tr>
<tr>
<td>1</td>
<td>non–corotational procedure is used</td>
</tr>
<tr>
<td>-1</td>
<td>approximate corotation (no rigid-body correction, the corotational projection operator is not used)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IMPTHE</th>
<th>0 – higher-order initial strain imperfections are used</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>only perturbed initial geometry is used</td>
</tr>
</tbody>
</table>
ICHIST  crack archive read option:
   0 – update crack status using archived data
   1 – use crack status and definitions from S1 only

IFLU  fluid-interaction option:
   0 – no fluid interaction
   1 – underwater shock (USA-STAGS) analysis — INACTIVE

ISOLVR  equation-solver-choice option:
   0 – assemble the matrix in STAGS’ SKYLINE format; use STAGS’ SKYLINE solver
   -1 – assemble the matrix system in STAGS’ compact format; use the VSS sparse matrix solver
   1 – specify the system-matrix format and the equation solver to use, plus additional control parameters, via the B-2 record, described next

NFABC  number of gradient fabrication specification records to process

\[
\begin{align*}
\text{if} \quad ( \text{ISOLVR} = 1 ) & \text{ then } \text{ go to B-2 } \\
\text{elseif} \quad ( \text{NFABC} > 0 ) & \text{ then } \text{ go to B-3 } \\
\text{else} & \text{ go to C-1 }
\end{align*}
\]
B-2 Solver Options

With this record, the user can specify the format in which system stiffness and stability matrices are to be assembled and can choose the equation solver to be used for the analysis to be performed.

With ISOLVR = 0 on the B-1 record, a future version of STAGS will choose the format and solver using basic system-size and/or solution-cost data; the current version of STAGS will use the program’s SKYLINE format and solver.

The following Solver-Options record is required if ISOLVR = 1 on record B-1:

---

ICPACT  IITER  IPRIM  IPRIS  ISAVE

<table>
<thead>
<tr>
<th>ICPACT</th>
<th>compact-format-selection parameter:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>use the SKYLINE system-matrix format and solver</td>
</tr>
<tr>
<td>&gt;0</td>
<td>use STAGS’ row-based COMPACT system-matrix format</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IITER</th>
<th>iterative-solution-method switch (ignored if ICPACT = 0):</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>use the (NASA/GSP) VSS compact-format equation solver</td>
</tr>
<tr>
<td>&gt;0</td>
<td>use STAGS’ (experimental) iterative solver</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IPRIM</th>
<th>assembled-matrix printout switch (ignored if ICPACT = 0):</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>do not print the assembled matrix</td>
</tr>
<tr>
<td>&gt;0</td>
<td>print the assembled matrix</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IPRIS</th>
<th>intermediate-solution-data printout switch (ignored if ICPACT = 0):</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>do not print intermediate solution data</td>
</tr>
<tr>
<td>&gt;0</td>
<td>print intermediate solution data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ISAVE</th>
<th>save-system-and-stop switch (ignored if ICPACT = 0):</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>proceed normally to obtain solutions within STAGS (default)</td>
</tr>
<tr>
<td>&gt;0</td>
<td>assemble the system matrix and RHS vector but do not solve the system within STAGS; save the matrix and RHS vector on ASCII-formatted output files, for examination and/or for solution operations outside STAGS), then stop</td>
</tr>
</tbody>
</table>

---

NFABC (B-1)

if ( NFABC > 0 ) then  go to B-3
else
  go to C-1
endif
B-3 Gradient Fabrication Specification Records

If and only if the \textit{NFABC} parameter is positive on record B-1, \textit{NFABC} records of this type are required to specify the beam, shell and/or solid fabrication(s) for which gradient computations are to be performed by the program. For more information about this process, please see Appendix E of this document.

\begin{itemize}
  \item \textbf{KFABTP} \textbf{KFB}
  \end{itemize}

- \textbf{KFABTP} fabrication-class specification parameter:
  \begin{itemize}
    \item -1 – beam-class GCP fabrication (not active in the current version of \textit{STAGS})
    \item -2 – shell-class GCP fabrication
    \item -3 – solid-class GCP fabrication
    \item 1 – beam-class wall fabrication
    \item 2 – shell-class wall fabrication
    \item 3 – solid-class wall fabrication
  \end{itemize}

- \textbf{KFAB} fabrication-identification parameter:
  \begin{itemize}
    \item 0 – use all fabrications of the specified class
    \item >0 – identifies a specific \textit{KFABTP}-class fabrication for which gradient computations are to be performed
  \end{itemize}

\begin{itemize}
  \item \textbf{NFABC} (B-1)
  \end{itemize}

\begin{verbatim}
if ( fewer than \textit{NFABC} records have been processed )then
  go to B-3
else
  go to C-1
endif
\end{verbatim}
Computational Strategy Parameters

The following is a summary of the input record sequence for each of the basic types of analysis that STAGS currently performs.

**Linear Analysis**
- C-1, C-5

**Eigenanalysis**
- C-1, C-5
- D-2, D-3

**Nonlinear Static Analysis**
- C-1, C-3, C-4, C-5
- D-1
- ET-1

**Transient Analysis**
- C-1, C-5
- D-1
- E-1–E-5

**Solution Branching**
- C-1, C-5
- D-1
- ET-1

**Load Relaxation**
- C-1, C-3, C-4, C-5
- D-1
- ET-1
C-1 Load Multipliers

This record is always included. The load factors specified here are meaningless with INDIC = 2 (B-1).

STAGS has two load systems, load system A and load system B. In each of these load systems, the actual load on the structure is equal to a load factor (or load multiplier) times a load distribution. For load system A, the load factor is called \( P_A \), and for load system B, the factor is called \( P_B \). The total load on the structure, \( L_{TOT} \), is equal to

\[
L_{TOT} = P_A \cdot L_A + P_B \cdot L_B
\]

where \( L_A \) and \( L_B \) are the two independent load distributions, which can be quite general; these distributions are determined by the Q records or by the user-written subroutines USRLD and UPRESS.

The first increments for each load system are determined by the load steps STEP(1) for system A and STEP(2) for system B. For arc-length solution control (see the discussion above), the first two load steps are performed using simple load control, after which the initial arc-length increments are adjusted based on convergence characteristics. The load increments can be positive, negative or zero, depending on the strategy chosen.

For a restart analysis under arc-length control, STEP(1) can be interpreted to mean arc length. If one selects NSTRAT < 0 (D-1), STEP(1) is applied as a scale factor to the arc length derived from the previous solutions. This is the variable labeled DETA in the OUT2 file, and can be found with the iteration summary that is always printed. For a continuation of the solution without interruption, set STEP(1) to unity, which means that the program will use the same DETA that it had on the last load step. The user is free to change STEP(1) to greater or less than unity to obtain better convergence.

A maximum load step is included for both load systems. Using this maximum, it is possible to “freeze” the load factor for either of the A and B systems at some level.

The initial load in a nonlinear analysis is STLD times the base load.
For linear analysis the total equals the initial load.

For bifurcation buckling analysis the critical load combination is the initial load for system B plus the eigenvalue times the initial load for system A.

For nonlinear static analysis the first load level at which solution is attempted is given by the initial load.

For arc-length (Riks) load control, load system B “floats” with load system A, according to the relation

\[ P_B = STLD(2) + \left\{ \frac{STEP(2)}{STEP(1)} \times [P_A - STLD(1)] \right\} \]

which makes the initial B load equal to \( STLD(2) \) and the next B load value \( STLD(1) + STLD(2) \) (see below), as would be the case with pure load control for the first two steps.

Boundary and loading conditions may be changed at restart. To avoid difficulties in converging to the new nonlinear solution state, the user should be careful to introduce such changes gradually.

In transient analysis (\( INDIC = 6 \) or 7), load parameters that are read on the E-2 record override the C-1 load factors. In order to permit the user to restart a transient analysis at the completion of a static analysis, the user can omit the time-dependent loading (E-2) and allow the C-1 factors to operate. For the most common application of a transient analysis restart at a fixed load level, \( STLD \) and \( FACM \) should be equal, and \( STEP \) should be zero for both load systems.
STLD(1)  STEP(1)  FACM(1)  STLD(2)  STEP(2)  FACM(2)  ITEMP  NFIX

STLD(1)  starting load factor for system A. * Not relevant for INDIC = 2 or 6 (see comments above).

STEP(1)  load factor increment for system A. The load increment is STEP(1) times the base load. Meaningful only for nonlinear analysis INDIC = 3, 4, 5, 6, or linear transient analysis, INDIC = 7 (B-1). If it is positive, the increment will be added to STLD(1); if negative, the absolute value of STEP will be subtracted from STLD. In an initial run under arc-length control, the solution will be executed under load control for the first two load steps, and the path parameter DETA will be adjusted according to the solution convergence characteristics. For load control restarts, STEP serves the same purpose as in the initial run. For arc-length control restarts, STEP will be a scale factor applied to the last path length step. For example, if STEP is 1.0, the previous path parameter value DETA will be taken from the restart file; if it is 0.5, half the previous DETA will be used. If a negative value of STEP is used, the solution will go in the reverse direction (or back up). It should be remembered that the previous path step may have corresponded to a decrease in the load factors.

FACM(1)  maximum load factor for system A. The maximum load is FACM times the base load (this is meaningful only for a nonlinear analysis $3 \leq \text{INDIC} \leq 5$). If \( \text{STLD}(1) < \text{FACM}(1) \), the run is terminated when the load exceeds \text{FACM}(1). If \( \text{STLD}(1) > \text{FACM}(1) \), the run is terminated when the load is less than \text{FACM}(1). If \text{FACM}(1) = \text{STLD}(1) \), the B load set takes over, even under arc-length control. In that case, solution stops only when $P_{g} \geq \text{FACM}(2)$.

STLD(2)  starting load factor for system B. For a creep analysis, (ICREEP = 1 on any I-1 record in the INP file), STLD(2) is the initial creep time.

STEP(2)  load factor increment for system B (meaningful only for nonlinear analysis $3 \leq \text{INDIC} \leq 5$). For a creep analysis, STEP(2) is the creep time increment.

* On restart, the value of this parameter must correspond to a solution saved from a previous run. (Compare ISTART, D-1).
FACM(2) maximum load factor for system B in a nonlinear analysis. For a creep analysis, this is the final creep time. When the B system load factor reaches FACM(2), it becomes fixed. Analysis continues until the A system has reached FACM(1). (But, see special use described under FACM(1).)

ITEMP thermal-loading option; user-written subroutine TEMP is required when ITEMP > 0.

0 – No thermal loads
1 – Thermal loading is generated and associated with load system A
2 – Thermal loading is generated and associated with load system B

NFIX if nonzero, the number of freedoms to be fixed, specified as
logical freedom numbers with the C-5 record

INDIC (B-1)

0 – linear static
1 – bifurcation buckling
2 – small vibrations (stress free state)
3 – nonlinear static
4 – nonlinear solutions at specified load levels
5 – small vibrations (linear or nonlinear stress state)
6 – transient response analysis (geometrically nonlinear)
7 – transient response analysis (geometrically linear)

if ( NFIX > 0 ) then
if ( INDIC=4 or INDIC=5 ) then go to C-3
else
  go to C-5
endif
elseif ( INDIC=0 ) then
  data deck is complete
elseif ( INDIC=1 or INDIC=2 ) then go to D-2
elseif ( INDIC=3 or INDIC=6 or INDIC=7 ) then go to D-1
elseif ( INDIC=4 or INDIC=5 ) then go to C-3
endif
This record is included only for bifurcation buckling analysis based on a nonlinear stress state or vibration analysis with initial stress, $\text{INDIC} = 4$ or $5$ (B-1).

Eigenvalues (bifurcation buckling load factor or vibration frequencies) and corresponding modes may be computed at a number of load steps. This choice allows use of boundary conditions and other constraints in the eigenvalue analysis that differs from those used in the analysis of the basic stress state. The load levels at which data are saved or used in eigenvalue analysis are specified in terms of the load factor corresponding to System A. Note that the deinstallation parameters are automatically selected for $\text{INDIC} = 4$ or $5$; therefore, the data on records D-2 and D-3 are not read in.

This record can also be used to extract solutions at specified load steps when using the arc-length control strategy. Since the load factors are part of the solution, there is no guarantee that a solution will be computed at particular desired load level. Setting $\text{INDIC} = 4$ solves the problem by switching to load control when near any load in the list (C-4).

---

### NLDS IXEV

<table>
<thead>
<tr>
<th>NLDS</th>
<th>IXEV</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of load factors for system A at which nonlinear stress states may be used as basis for an eigenvalue analysis (directly or in a subsequent run), $\text{NLDS} \leq 20$; if $\text{NLDS} = -1$, then the load factor list will be generated automatically by STAGS using data on the C-4 record.</td>
<td>0</td>
<td>do not execute eigenvalue solution</td>
</tr>
<tr>
<td>$&gt;0$</td>
<td>number of modes desired</td>
<td></td>
</tr>
</tbody>
</table>

$go to$ C-4
C-4 Load Factors

This record is included only if INDIC = 4 or 5 (B-1). It gives the value of the load factor at which eigenvalue solutions are desired (buckling loads or vibration frequencies). The automatic choice of step size in the nonlinear analysis is overridden as necessary so that solutions are obtained for the specific set of NLDS values (C-3). The critical load factor in the case of bifurcation buckling from a nonlinear stress state is computed for load factor i as $\text{PLDS}(i) \times (1 + \lambda)$ where $\lambda$ is the eigenvalue defined in the output. The result is rigorously valid only when the computed eigenvalue equals zero. This option works with arc-length solution control as well.

$\text{PLDS}(i)$, $i = 1$, NLDS

PLDS(i) /\text{load factor values for system A at which nonlinear prestress solutions are desired; may be used as basis for an eigenvalue analysis (directly or in a subsequent run)}\

If $\text{NLDS} = -1$ on the C-3 record, then read three variables (PLDA, PLDB and PLDSTEP) on the C-4 record, where PLDA is the starting load factor, PLDB is the ending load factor and PLDSTEP is the desired load step size between these values. The number of solution steps is NLDS, which is defined as

$$\text{NLDS} = \text{NINT} \left[ \frac{\text{PLDB} - \text{PLDA}}{\text{PLDSTEP}} \right]$$

This option is useful to obtain a series of solutions at evenly-spaced load steps over all or only a portion of the solution space. The load factor may be adjusted due to convergence difficulties.

Note: Because of the possible complexity of load paths when using arc-length control, it is important to choose the list of load factors in the order expected for the problem at hand. Such a choice may become complicated for strongly-nonlinear cases.
C-5 Suppress Selected Freedoms

The C-5 record provides the user with a mechanism to suppress nonstandard freedoms such as Lagrange constraints and other specialized freedoms that are generated automatically by the STAGS code. These freedoms are in contrast to ordinary nodal freedoms that are defined by the user and known in advance. Any type of freedom can be suppressed in the model input data with Q and/or U records. Sometimes, however, specialized freedoms turn out to be singular or ill-conditioned during system-matrix decomposition. This happens most often when Lagrange constraints are found to be redundant—when partial compatibilities and/or other constraints are also applied for the same freedoms.

If STAGS detects that any freedom is singular, it prints an informative message containing the logical freedom number and the model freedom designation for that freedom. If that freedom is a translational freedom, execution is stopped (because the user has not restrained rigid-body motion). If it is a rotational or a specialized freedom, that freedom is suppressed for the remainder of the current run and any subsequent continuation runs. Under some circumstances (when planar boundary conditions are applied to a not-quite-planar boundary, for example) this automatic deletion process fails, leading to erroneous results. Fortunately, STAGS also prints informative messages when any freedom is found to be ill-conditioned during the decomposition process. This provides the user with the information that is required to revise the model-definition input or to use this C-5-record mechanism in the solution phase to suppress those freedoms in subsequent runs.

**IFIX(i), i = 1, NFIX**

<table>
<thead>
<tr>
<th>IFIX(i)</th>
<th>logical freedoms to be suppressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>linear static</td>
</tr>
<tr>
<td>1</td>
<td>bifurcation buckling</td>
</tr>
<tr>
<td>2</td>
<td>small vibrations (stress free state)</td>
</tr>
<tr>
<td>3</td>
<td>nonlinear static</td>
</tr>
<tr>
<td>4</td>
<td>nonlinear solutions at specified load levels</td>
</tr>
<tr>
<td>5</td>
<td>small vibrations (linear or nonlinear stress state)</td>
</tr>
<tr>
<td>6</td>
<td>transient response analysis (geometrically nonlinear)</td>
</tr>
<tr>
<td>7</td>
<td>transient response analysis (geometrically linear)</td>
</tr>
</tbody>
</table>

if ( INDIC=0 ) then  
data deck is complete
elseif ( INDIC=1 or INDIC=2 ) then  
go to D-2
else  
go to D-1
endif
D-1 Strategy Parameters

This record is included only for nonlinear analysis (including nonlinear stress states for bifurcation buckling or vibration analysis) and transient response analysis, \(3 \leq \text{INDIC} \leq 6\) (B-1).

For nonlinear analysis and transient response analysis, intermediate data can be saved on the file `casename.rst`, and retrieved in a subsequent run so that the analysis can be restarted and continued. If \(\text{IPOST} = 0\) (B-1), the last 3 displacement solutions obtained in a run are saved on file `casename.rst` to provide the solution algorithm with enough data to extrapolate to the next solution state, just as if the solution were continuing without interruption. If \(\text{IPOST} > 0\), displacements are saved every \(\text{IPOST}, \text{IPOST} - 1\), and \(\text{IPOST} - 2\) steps, which means that if \(\text{IPOST} \leq 3\), all displacements are saved. For an example where \(\text{IPOST} > 3\), let \(\text{IPOST} = 10\); then steps 8, 9, and 10; 18, 19, and 20; and so on are saved. For this case a smooth restart can be attempted at steps 10, 20, and so on.

When the analysis is restarted, it is important that the starting load factors (C-1) correspond to the displacement on the record from which restart is attempted.

If convergence difficulties are encountered during the nonlinear iteration, STAGS will form and factor the stiffness matrix before continuing the iterations. The variable \(\text{NEWT}\) sets an upper limit on the number of refactorings allowed. If \(\text{NEWT}\) is set less than zero, refactoring is performed on every \(i\)th load step, where \(i = |\text{NEWT}|\), no matter how quickly the iterations lead to convergence, with no limit on the number of refactorings. If \(\text{NEWT} = -20\), the true Newton method is used (refactoring on all iterations). This method can lead to accelerated convergence rates, but it can be very costly (see the introduction to this section for more detail).

If convergence difficulties persist, the increment that controls the arc length or load step will be cut. The size of this cut is a function of the iteration history leading up to the convergence failure. The variable \(\text{NCUT}\) puts an upper limit on how often the program can cut the step. Without this limit, the program can labor endlessly with unreasonably small increments; when \(\text{NCUT}\) is exceeded, the program exits gracefully, saving data for restart. When this happens, the user should examine the model and its loading environment carefully for important changes in the nature of the response. It is often necessary to initiate a bifurcation analysis at this point and switch to an alternate solution path.

Alternately, it may be necessary to initiate a new analysis, with a more refined imperfection pattern, or to switch to a transient analysis strategy (E-1–E-5).

The solution control strategy for static analysis is set by the variable \(\text{NSTRAT}\). If \(\text{NSTRAT} = 0\) (default), then arc-length control takes effect after the load step \(\text{STLD} + \text{STEP}\) has been computed under load control. Thus, for the initial run, two solutions are computed under load
control: \textbf{STLD} and \textbf{STLD + STEP}. These variables are found on record C-1, with the B load computed according to the formula on page 8-12. For a restart run, only the solution \textbf{STLD + STEP} is computed, because the solution for \textbf{STLD} already exists. All subsequent solutions are computed using arc-length control. For an initial run with \textbf{NSTRAT} = -1, the result is the same as for \textbf{NSTRAT} = 0. For a restart, however, the user input values of \textbf{STLD(1)} and \textbf{STLD(2)} (C-1) are checked with those stored with the restart step \textbf{ISTART} (see below). If they are different, execution terminates with an error message. If they are the same to within a very small tolerance, the \textbf{STEP(1)} parameter (C-1) is interpreted as a scale factor applied to the arc-length parameter \textbf{DETA} (C-1). For an ordinary restart with continuity expected from step \textbf{ISTART} to \textbf{ISTART} + 1, this is the most efficient mode of operation. If \textbf{NSTRAT} = 1, all subsequent steps are executed under load control.

The error tolerance for convergence (\textbf{DELX}) gives the error tolerance of the residual norm or displacement norm, whichever is larger. The residual error norm is normalized to the applied load and reaction forces from the initial solution step. If \textbf{DELX} is set to zero or left blank, a default value of 0.001 is used. If \textbf{WUND} is zero or blank, an initial relaxation of 1.0 is used. The relaxation factor is increased (automatically), if convergence is monotonic but slow, and it is decreased if convergence is oscillatory. If a nonzero value of \textbf{WUND} is specified, this value will be used regardless of the convergence behavior. It is uncommon to reset \textbf{WUND}.

Sometimes, especially during adaptive refinement or during a crack growth simulation, other software will generate a new mesh (essentially a different model) of the same physical system. In that case, it is advantageous to estimate the displacements at the nodal positions using interpolation from the previous converged solution. It is well known that these displacements, although not satisfying equilibrium for the new mesh, may in fact be an excellent initial guess for a nonlinear iteration to a subsequent solution and continuation with the new model. A special provision has been provided for that case. A \textbf{STAR} routine has been provided to place the initial guess displacement field into the database with a loadstep number, \textit{n}. \textit{n} is selected for the user’s convenience. To tell \textbf{STAGS} to use these displacements instead of an ordinary restart, choose \textbf{ISTART} = \textit{-n} below. An error will be reported if the special vector has not been installed in the \textbf{STAGS} database \textit{via} \textbf{STAR}. 

\begin{align*}
\text{control: } \textbf{STLD} \text{ and } \textbf{STLD + STEP}. \text{ These variables are found on record C-1, with the B load computed according to the formula on page 8-12. For a restart run, only the solution } \textbf{STLD + STEP} \text{ is computed, because the solution for } \textbf{STLD} \text{ already exists. All subsequent solutions are computed using arc-length control. For an initial run with } \textbf{NSTRAT} = -1, \text{ the result is the same as for } \textbf{NSTRAT} = 0. \text{ For a restart, however, the user input values of } \textbf{STLD(1)} \text{ and } \textbf{STLD(2)} \text{ (C-1) are checked with those stored with the restart step } \textbf{ISTART} \text{ (see below). If they are different, execution terminates with an error message. If they are the same to within a very small tolerance, the } \textbf{STEP(1)} \text{ parameter (C-1) is interpreted as a scale factor applied to the arc-length parameter } \textbf{DETA} \text{ (C-1). For an ordinary restart with continuity expected from step } \textbf{ISTART} \text{ to } \textbf{ISTART} + 1, \text{ this is the most efficient mode of operation. If } \textbf{NSTRAT} = 1, \text{ all subsequent steps are executed under load control.}
\end{align*}

The error tolerance for convergence (\textbf{DELX}) gives the error tolerance of the residual norm or displacement norm, whichever is larger. The residual error norm is normalized to the applied load and reaction forces from the initial solution step. If \textbf{DELX} is set to zero or left blank, a default value of 0.001 is used. If \textbf{WUND} is zero or blank, an initial relaxation of 1.0 is used. The relaxation factor is increased (automatically), if convergence is monotonic but slow, and it is decreased if convergence is oscillatory. If a nonzero value of \textbf{WUND} is specified, this value will be used regardless of the convergence behavior. It is uncommon to reset \textbf{WUND}.

Sometimes, especially during adaptive refinement or during a crack growth simulation, other software will generate a new mesh (essentially a different model) of the same physical system. In that case, it is advantageous to estimate the displacements at the nodal positions using interpolation from the previous converged solution. It is well known that these displacements, although not satisfying equilibrium for the new mesh, may in fact be an excellent initial guess for a nonlinear iteration to a subsequent solution and continuation with the new model. A special provision has been provided for that case. A \textbf{STAR} routine has been provided to place the initial guess displacement field into the database with a loadstep number, \textit{n}. \textit{n} is selected for the user’s convenience. To tell \textbf{STAGS} to use these displacements instead of an ordinary restart, choose \textbf{ISTART} = \textit{-n} below. An error will be reported if the special vector has not been installed in the \textbf{STAGS} database \textit{via} \textbf{STAR}. 

\begin{align*}
\text{The error tolerance for convergence (} \textbf{DELX} \text{) gives the error tolerance of the residual norm or displacement norm, whichever is larger. The residual error norm is normalized to the applied load and reaction forces from the initial solution step. If } \textbf{DELX} \text{ is set to zero or left blank, a default value of 0.001 is used. If } \textbf{WUND} \text{ is zero or blank, an initial relaxation of 1.0 is used. The relaxation factor is increased (automatically), if convergence is monotonic but slow, and it is decreased if convergence is oscillatory. If a nonzero value of } \textbf{WUND} \text{ is specified, this value will be used regardless of the convergence behavior. It is uncommon to reset } \textbf{WUND}.
\end{align*}

\begin{align*}
\text{Sometimes, especially during adaptive refinement or during a crack growth simulation, other software will generate a new mesh (essentially a different model) of the same physical system. In that case, it is advantageous to estimate the displacements at the nodal positions using interpolation from the previous converged solution. It is well known that these displacements, although not satisfying equilibrium for the new mesh, may in fact be an excellent initial guess for a nonlinear iteration to a subsequent solution and continuation with the new model. A special provision has been provided for that case. A } \textbf{STAR} \text{ routine has been provided to place the initial guess displacement field into the database with a loadstep number, } \textit{n}. \textit{n} \text{ is selected for the user’s convenience. To tell } \textbf{STAGS} \text{ to use these displacements instead of an ordinary restart, choose } \textbf{ISTART} = \textit{-n} \text{ below. An error will be reported if the special vector has not been installed in the } \textbf{STAGS} \text{ database } \textit{via} \textbf{STAR}.
\end{align*}
Solution Input Summary and Control Parameters

ISTART NSEC NCUT NEWT NSTRAT DELX WUND

ISTART starting code:

0 – begin new case

n – a positive integer specifies restart from the nth load step. Make sure that corresponding data have been saved (see IPOST, B-1). n must correspond to the same load step as STLD (C-1).

-n – a negative integer specifies restart from the special vector identified by the positive integer n, where vector n was installed in the STAGS database via STAR; used most commonly during nonlinear adaptive refinement or crack growth simulation.

NSEC number of CPU seconds at which run will be terminated and data saved on restart file; NSEC = 0 permits unlimited time

NCUT total number of times the step size may be cut

NEWT total number of refactorings allowed; use of NEWT < 0 is discussed above

NSTRAT 0 – execute the first step(s) under load control, and then switch to arc-length control (see discussion above)

-1 – for initial run, proceed as for NSTRAT = 0; otherwise, interpret STEP(1) (C-1) as a scale factor for the arc-length parameter DETA. See the C-1 description for more complete information.

1 – execute under load control

NSTRAT is ignored for a transient analysis

DELT error tolerance; the default value for DELX is 0.001

WUND relaxation factor; the default value is 0.001

INDIC (B-1)

6 – transient response analysis (geometrically nonlinear)

7 – transient response analysis (geometrically linear)

if (INDIC=6 or INDIC=7) then go to E-1
else go to ET-1
**D-2 Eigenvalue Control**

The record is included only for *linear* buckling or vibration analysis.

If **DELEV** is zero or blank, a default value of 0.00001 is used. When the CPU time limit **NSEC** is reached, current estimates for eigenvalues and modes are printed and saved on the restart file as specified by **IPRINT** even if the convergence criterion is not satisfied. Use **IPRINT** = 0 or 1 unless there is a good reason to suppress the extra output. Eigenmodes are also saved on a special *casename.imp* file for possible use as initial bucking imperfections on a subsequent run.

<table>
<thead>
<tr>
<th>NSEC</th>
<th>DELEV</th>
<th>IPRINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of CPU seconds at which eigenvalue analysis will be terminated; <strong>NSEC</strong> = 0 permits unlimited time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>eigenvalue error tolerance; the default value for <strong>DELEV</strong> is 0.001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 – print eigenvalues, modes and intermediate iteration data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 – print eigenvalues and intermediate iteration data; suppress printout of the modes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 – print the eigenvalues and modes, but suppress printout of intermediate iteration data (which are not bulky and may provide useful approximate information about eigenvalues beyond those requested on the D-3 record)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 – print eigenvalues only</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*go to D-3*
D-3 Cluster Definition

If $\text{EIGA} = \text{EIGB}$, a number (NEIG) of eigenvalues closest to (above or below) a specified value (SHIFT) will be determined. If $\text{EIGA} < \text{EIGB}$, SHIFT is ignored and the NEIG eigenvalues closest to the center of the interval ($\text{EIGA}$, $\text{EIGB}$) are determined. Computations terminate when all eigenvalues in the interval have been found, even if their number is less than NEIG.

For buckling problems, INDIC = 1 or 4 (B-1), SHIFT, EIGA, EIGB relate to critical load factors (the load factors at which the Jacobian determinant vanishes). For vibration problems (INDIC = 2 or 5, they relate to frequencies (in cps). In this case (vibration analysis) the “center” $c$ of the interval ($\text{EIGA}$, $\text{EIGB}$) is chosen to satisfy: $c^2 = (\text{EIGA}^2 + \text{EIGB}^2)/2$

If rigid body modes are to be included in a vibration analysis (INDIC = 2 or 5), an eigenvalue shift must be used (SHIFT, D-3) to prevent matrix singularities.

<table>
<thead>
<tr>
<th>NEIG</th>
<th>SHIFT</th>
<th>EIGA</th>
<th>EIGB</th>
</tr>
</thead>
<tbody>
<tr>
<td>maximum number of eigenvalues to be computed, $\text{NEIG} \leq 20$</td>
<td>initial eigenvalue shift; for vibration analysis of a free body, it is necessary to specify a non-zero shift</td>
<td>lower bound of eigenvalue range</td>
<td>upper bound of eigenvalue range</td>
</tr>
</tbody>
</table>

* data deck is complete
E-1 Time Integration—Record 1

The E-1 and E-2 records are included only for transient analysis, when INDIC = 6 or 7 (B-1). The user has the option of using a fixed or an automatically-controlled variable time step (IERRF, E-2). A maximum expected value of the displacement SUP is used in connection with the automatic step size control. Load systems A and B are used to define two independent loading histories, analogous to the static analysis option.

In the equation $M\ddot{u} + D(\dot{u}) + S(u) = f(t)$, $D$ represents a damping vector. Its elements are determined by $D = \alpha M\ddot{u} + \beta \dot{S} + \gamma P\dot{u}$. The diagonal matrix $\gamma P$ represents external, velocity dependent forces. The velocity-dependent base loads are defined in the sequence Q-1 through Q-3 in the same way as the regular forces on the structure. The damping matrix is obtained through multiplication by the “damping factor” $\gamma$.

<table>
<thead>
<tr>
<th>TMIN</th>
<th>TMAX</th>
<th>DT</th>
<th>SUP</th>
<th>ALPHA</th>
<th>BETA</th>
<th>GAMMA</th>
<th>THOLD</th>
</tr>
</thead>
<tbody>
<tr>
<td>time, $t$, at which transient analysis begins</td>
<td>time, $t$, at which transient analysis terminates; $\text{TMIN} \leq \text{TMAX}$</td>
<td>time step; initial time step if variable step selected (IERRF, E-2)</td>
<td>maximum expected displacement; relevant only with variable time step</td>
<td>mass damping factor $\alpha$</td>
<td>stiffness damping factor $\beta$</td>
<td>damping factor $\gamma$, which multiplies the velocity-dependent forces to give the “external damping” — INACTIVE</td>
<td>the value of the problem time at which changes of the time step first are allowed. THOLD can be used to suppress automatic time step increases during the initial phase so that the loading history may be accurately defined.</td>
</tr>
</tbody>
</table>

NOTE: The temporal variation of load factors $(P_A, P_B)$ is defined on E-2 as $P_A = f(t)\cdot P_A \quad P_B = g(t)\cdot P_B$, where $f$ and $g$ can have one of several standard forms (see Figure 11.1 on page 11-28). Also note that a transient analysis can be restarted and that the load-factor history can be defined independently for a restart. Basic loading on the structure can also be changed for restart.


go to E-2
E-2 Time Integration—Record 2

Include only for transient analysis INDIC = 6 or 7 (B-1).

With simple input data, a load-factor history is independently-defined for each of the load systems A and B. The temporal variation of load factors \( P_A, P_B \) is defined \( P_A = f(t), P_B = g(t) \), where \( t \) is time, and \( f \) and \( g \) are of the forms illustrated in Figure 11.1 on page 11-28. Note also that a transient analysis can be restarted and that the load-factor history can be defined independently for a restart. Basic loading on the structure can also be changed for restart. As the examples below demonstrate, clever definition of the A/B load histories, combined with effective restarting strategies, provides tremendous power and flexibility for applying arbitrary transient loading.

A “box wave,” for example, can be obtained from the linearly-varying load-factor history case (a) by setting

\[ CA2 = CA3 = 0 \quad CA4 = CA5 \]

Trigonometric variation is specified by load-factor history case (b) according to

\[ P = A \cos \alpha + C \quad \alpha = \frac{2\pi}{T} (t - \phi) \quad (11.3) \]

where \( A \) is the amplitude, \( T \) is the period, \( \phi \) is a phase shift, and \( C \) is a constant. The half-period, \( p \), is defined as \( p = T/2 \), and STAGS input describing (11.3) is

\[ CA1 = A \quad CA2 = C \quad CA3 = p \quad CA4 = \phi \]

Therefore, (11.3) can be rewritten in terms of input data as

\[ P = CA1 \cdot \cos \left( \frac{2\pi}{2 \cdot CA3} (t - CA4) \right) + CA2 \]

A cosine function is defined by

\[ CA1 = A \quad CA2 = 0 \quad CA3 = p \quad CA4 = 0 \]

To define a sine function, the cosine function is shifted by 1/4 period, \( T/4 = p/2 \). Therefore, a sine function is defined as

\[ CA1 = A \quad CA2 = 0 \quad CA3 = p \quad CA4 = p/2 \]
Though not readily apparent, (11.3) can be used to specify a \( \cos^2 \) (cosine squared) function defined by

\[
P = A \cos^2 \alpha + C \\
\alpha = \frac{\pi}{T}(t - \varphi)
\]

where \( A \) is the \( \cos^2 \) amplitude, and \( T \) is the \( \cos^2 \) period. By noting that \( \cos^2 \) is periodic in \( \pi \), rather than \( 2\pi \); and that the range of \( \cos^2 \) is \([0, A]\), rather than \([-A, A]\); and by taking advantage of the trigonometric power relation \( \cos^2 \alpha = 1/2(\cos 2\alpha + 1) \); it follows that (11.3) defines a \( \cos^2 \) function when

\[
CA1 = A/2 \quad CA2 = A/2 \quad CA3 = p \quad CA4 = 0
\]

For each of the load-factor history types, the duration of the corresponding loading event is determined by specification of both an initial time, \( t_i \), and a final time, \( t_f \); thus, the domain of \( t \) is \( t_i \leq t \leq t_f \). Referring to (11.3) and Figure 11.1, the duration of the loading event represented by load-factor history case (b) is defined by

\[
CA5 = t_i \quad CA6 = t_f
\]

For example, a single-wave cosine history starting at time \( t = 0 \) and ending at time \( t = T \) is defined by

\[
CA1 = A \quad CA2 = 0 \quad CA3 = p \quad CA4 = 0 \quad CA5 = 0 \quad CA6 = T
\]

Note that the initial/final times for load-factor histories are defined independently of the starting/stopping times for the transient analysis (TMIN/TMAX, E-1). For example, in load-factor history case (b), if \( T \text{MIN} < CA5 \), the structure is unloaded for \( t < CA5 \). If \( CA6 < T \text{MAX} \), the structure is unloaded for \( t > CA6 \).

Additional examples follow to illustrate the flexibility provided by creative definition of load-factor histories together with judicious combination of histories for the A/B systems. Keeping in mind that a transient analysis can be restarted and that the load-factor history, as well as the basic loading on the structure, can be defined independently for a restart, it can be seen that transient-loading capabilities are quite general in STAGS.

An impulse consisting of a single-wave \( \sin^2 \) (sine squared) form with an amplitude of \( A \) and a duration of one period, \( T \), starting at time \( t = 0 \) at a load factor of \( P = 0 \), increasing to \( P = A \) at \( t = T/2 = p \), then falling back to \( P = 0 \) at \( t = T \), is obtained from case (b) with:

\[
CA1 = A/2 \quad CA2 = A/2 \quad CA3 = p \quad CA4 = p \quad CA5 = 0 \quad CA6 = T
\]

Note that \( \cos^2 \) is shifted by one half-period, \( p \), to obtain \( \sin^2 \).
The load factor may be increased from \( P_0 = 0 \) at \( t = 0 \) to a value of \( P = A \) at \( t = t_1 \) with a \( \sin^2 \) form, and then held constant if \( p_A \) is determined from case (b) with:

\[
\begin{align*}
CA_1 &= A/2 \\
CA_2 &= A/2 \\
CA_3 &= t_1 \\
CA_4 &= t_1 \\
CA_5 &= 0 \\
CA_6 &= t_1
\end{align*}
\]

and \( p_B \) is determined from case (a) with:

\[
\begin{align*}
CA_1 &= A \\
CA_2 &= CA_3 = t_1 \\
CA_4 &= CA_5 = TMAX
\end{align*}
\]

The load factor may be linearly increased from \( P_0 = 0 \) at \( t = 0 \) to a value of \( P = A \) at \( t = t_1 \), and then exponentially decayed to \( P = A/2 \) at \( t = t_2 \) if \( p_A \) is determined from case (a) with:

\[
\begin{align*}
CA_1 &= A \\
CA_2 &= 0 \\
CA_3 &= CA_4 = CA_5 = t_1
\end{align*}
\]

and \( p_B \) is determined from case (c) with:

\[
\begin{align*}
CA_1 &= A \\
CA_2 &= t_1 \\
CA_3 &= t_2 - t_1 \\
CA_4 &= A/2
\end{align*}
\]
IMPL METHOD IERRF IVELO IFORCE IPA IPB

IMPL  
0 – implicit integration
1 – explicit integration

METHOD  
indicates type of implicit integration
1 – trapezoidal formula
2 – Gear’s second-order formula INACTIVE
3 – Gear’s third-order formula INACTIVE
4 – K. C. Park’s formula*

IERRF  
0 – constant time step
1 – variable (automatically-controlled) time step

IVELO  
number of weighted modal velocities, defined on E-5 records

IFORCE  
0 – no user written FORCET
1 – user written FORCET

IPA  
load-factor history option for system A; see Figure 11.1
0 – load system A not included
1 – linear variation, case (a)
2 – trigonometric variation, case (b)
3 – exponential decay, case (c)

IPB  
load-factor history option for system B; see Figure 11.1
0 – load system B not included
1 – linear variation, case (a)
2 – trigonometric variation, case (b)
3 – exponential decay, case (c)

if (IPA > 0) then go to E-3
elseif (IPB > 0) then go to E-4
elseif (IVELO > 0) then go to E-5
else data deck is complete

Case (a) Linear variation of load factor

Case (b) Trigonometric variation of load factor

Case (c) Exponential decay of load factor

Figure 11.1 Load-factor histories.
E-3 Load History—Record 1

This record defines constants to be used in the definition of the load-factor history corresponding to system A. It is only included if $\text{IPA} > 0$ (E-2).

\begin{tabular}{llllll}
CA1 & CA2 & CA3 & CA4 & CA5 & CA6 \\
\end{tabular}

variables defined in Figure 11.1.

- IPB (E-2) load factor variation for system B
- IVELO (E-2) number of weighted modal velocities

if $(\text{IPB} > 0)$ then \textit{go to E-4}
elseif $(\text{IVELO} > 0)$ then \textit{go to E-5}
else \textit{data deck is complete}
**E-4 Load History—Record 2**

This record defines constants to be used in the definition of the load-factor history corresponding to system B. It is only included if $IPB > 0$ (E-2).

<table>
<thead>
<tr>
<th>CB1</th>
<th>CB2</th>
<th>CB3</th>
<th>CB4</th>
<th>CB5</th>
<th>CB6</th>
</tr>
</thead>
</table>

CB1, CB2, CB3, CB4, CB5, CB6 variables defined in Figure 8.1

**IVELO (E-2)** number of weighted modal velocities

if $IVELO > 0$ then go to E-5
else data deck is complete
## E-5 Weighted Modal Initial Velocity

These records are included only when specified by setting \texttt{IVELO} $> 0$ (E-2). \texttt{STAGS} is set up to allow the user to include initial velocities from a number of executions of the same case. For example, one may have initially performed a linear bifurcation analysis, followed by a nonlinear collapse analysis, followed by an eigenanalysis base upon a nonlinear stress state. Another execution could be run from the beginning, including all the modes accumulated so far. \texttt{STAGS} always produces two files, the \texttt{IMP} file (\texttt{casename.imp}) and the \texttt{EGV} file (\texttt{casename.egv}), that contain all the modes computed during the entire history of the case. Each mode is identified by a unique combination of values for the data \texttt{IMSTEP}, \texttt{IMMODE}, and \texttt{IMRUN}. These data are written to the \texttt{STAGS} database when modes are saved. The user can generate an initial velocity field from a linear combination of these modes, specifying amplitudes with \texttt{EIGA}.

Remember, for each of the eigenmodes that is computed by \texttt{STAGS}, the largest \textit{translational} component is normalized to unity.

### EIGA IMSTEP IMMODE IMRUN

<table>
<thead>
<tr>
<th>EIGA</th>
<th>buckling mode amplitude; may be $&lt; 0$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMSTEP</td>
<td>load step number</td>
</tr>
<tr>
<td>IMMODE</td>
<td>mode number</td>
</tr>
<tr>
<td>IMRUN</td>
<td>eigenvalue-execution ID number</td>
</tr>
</tbody>
</table>

\texttt{IVELO} (E-2) number of weighted modal velocities

```plaintext
if (IVELO modes have been defined) then
data deck is complete
else
continue defining  E-5
endif
```
ET-1 Solution Control

The most common choice for the variable NPATH (below) is 0.

If NSTRAT = 1 (D-1), load control is used throughout the analysis. If NSTRAT < 1, the path parameter method is used except for special starting or restarting procedures (which may also be affected by NSOL, ET-1).

For all values of NSTRAT and NSOL, the starting procedure (ISTART = 0, D-1) is identical. The linear solution with load factors $P_A$ and $P_B$ is computed as step 0 and saved on file—if IPOST1 > 0 (B-1). A non-linear solution is then obtained with the same load factors and saved as step 1. The second non-linear solution, step 2, is computed with values of $P_A$ and $P_B$ incremented by the load increments on the C-1 record. For NSTRAT = 1, the remaining analysis continues under load control. However if NSTRAT < 1, the solution procedure shifts to the path parameter method using the initial two nonlinear solutions to establish a path.

Restarting procedures are fundamentally more complex. The input variable NSOL (ET-1) is introduced to provide greater flexibility. NSOL = 0 implies that restart is continuous with respect to loading and boundary conditions and therefore solutions stored on the restart file may be used in determining initial values for the next solution. If NSOL = 1, the solution is assumed discontinuous and only the restart step is used in the new solution. In either case, the restart step may not be in equilibrium because of changes in input loading or boundary conditions. Significant difficulties in convergence behavior could result from this situation.

To help the user, the program evaluates equilibrium for the restart step and prints a warning message when a serious state of non-equilibrium is noted. An experimental option NSOL = 2 is being introduced to take special measures when the equilibrium test fails; more will be said about this option after further testing. If one is far from equilibrium and convergence fails, the relaxation procedure (NPATH = 5, ET-1) should be used first to return to equilibrium. A description of this option is found below.

If NSTRAT = 0, restart begins with one step under load control; values of $P_A$ and $P_B$ are incremented by STEP (C-1). The solution procedure then reverts to the path parameter method for the remainder of the analysis. The operation of the load incrementation algorithm is also described in record C-1.

If NSTRAT < 0, restart begins as a continuation of the path procedure. This requires the existence of at least two solutions on the restart file. The STEP increment on record C-1 is then interpreted as a path increment factor. This factor scales the new path step based on the last path length on the restart file. Thus if STEP = 1.0, the path continues with the same arc length used in the restart. The increment in path length may be doubled or halved, etc. as seems desirable. The value of STEP may also be negative which would reverse the path direction for restart. This is a convenient method for difficult unloading situations such as...
elastic unloading from a state of plastic deformation. After the first new solution is obtained, the analysis continues with the standard path parameter method. See “C-1 Load Multipliers” on page 11-11.

A special option NPATH = 1 is provided that switches the solution to an experimental “tangent” Equivalence Transformation method. Although this option helps in some cases, it is not recommended for ordinary analyses. The options NPATH = 2 and NPATH = 3 are the Equivalence Transformation options for branching on bifurcation. These options are described in Section 11.3.

The option NPATH = 5 allows for “Load Relaxation” in which the solution is to be relaxed from a nonequilibrium state. This is the only option provided for doing this. If NPATH = 5 (and INDIC = 3, B-1). Although ISTART (on D-1), and STLD(1) and STLD(2) (on C-1) govern the load state where the new equilibrium will be computed, the remainder of the information on the C-1 record controls the behavior after load relaxation has been completed, just as in a regular nonlinear run. This means that STAGS first executes the load relaxation process, and upon successful convergence to a new equilibrium state, continues with the execution after load relaxation is complete, automatically producing new solutions as directed by the user. The program internally determines what load relaxation parameters to use based on an estimate of how far from equilibrium the current state is. The interested reader is encouraged to obtain the references cited in the footnote.*

A special option NPATH = -1 has been provided to use the solution at restart to define the nonlinear stress state, stiffness and stability matrices for a buckling analysis. No new solutions are computed, and the analysis terminates after NEV eigenmodes have been found. Restart boundary conditions and load values can be altered as in any other restart. The ability to change boundary conditions offers the user a capability similar to the alternate buckling boundary conditions permitted for a linear bifurcation.

* See
and
**Solution Input Summary and Control Parameters**

### NPAT\textbf{H} NE\textbf{V} NS\textbf{OL} IE IGNORE LDMAX IU\textbf{P}LDA IU\textbf{P}LDB

**NPAT\textbf{H}**

0  –  use option chosen by **NSTRAT** (D-1)
1  –  use the “tangent” **ET** method (experimental method)
-1  –  compute **NEV** buckling modes on restart at load step **ISTART** (D-1), and PA and PB loads at **STLD**(1) and **STLD**(2) (C-1), respectively
2  –  single mode branch switching (see 11.3 on page 11-36)
3  –  special ET option, see next section — INACTIVE
5  –  load relaxation (see explanation above)

**NE\textbf{V}**

0  –  do not compute eigenmodes at run termination
>0  –  compute **NEV** eigenmodes at run termination

**NS\textbf{OL}**

0  –  assume the solution is continuous on restart, and perform quadratic extrapolation to estimate a new solution wherever possible
1  –  assume that the solution is *discontinuous*, and do not extrapolate to estimate the next solution. In this case, the previous solution is used as the initial estimate, and the solution will be checked for equilibrium. If equilibrium is satisfied, proceed to full extrapolation after enough solutions have been accumulated. If equilibrium is not satisfied, print a warning message and attempt to find a new solution. If convergence is obtained, proceed to full extrapolation after enough solutions have been accumulated.
2  –  special experimental option under development
3  –  use the previous solution vector to start the first two iterations of each solution step; this option is particularly useful during progressive failure analysis.

Please see the next section for the role **NSOL** plays for the **ET** Bifurcation Processor special options

**IE**

mode number for branch switching. Used only for special ET Bifurcation Processor options described in the next section.

**IGNORE**

ignored; included for compatibility with old *casename.bin* files

**LDMAX**

0  –  continue solution beyond any limit points found (points of maximum load, load system A only)
1  –  stop solution upon reaching a maximum load (load system A only)
IUPLDA if nonzero, interpret the A load system as *follower loads*, or loads that *rotate* with each node

IUPLDB if nonzero, interpret the B load system as *follower loads*, or loads that *rotate* with each node

* data deck is complete
11.3 The Equivalence Transformation Bifurcation Processor (ET)

The Equivalence Transformation bifurcation processor (ET) provides a means to branch to alternate solution paths during the analysis of structures well into the postbuckling regime*. In many cases routine analysis fails because the equation system becomes ill conditioned near the critical limit and bifurcation points. Whereas path continuation methods offer help for limit point analyses, bifurcation points have proved more difficult. ET allows the user to select a solution branch in the direction of growth of a bifurcation mode that has been computed for the nonlinear stress state near the critical point. Combined with the usual arc-length control algorithms, the user can in principle turn from one solution branch to the next as part of an investigation of the solution space of lowest energy most likely to be seen in physical systems.

In order to use the ET options, the user must first decide very carefully what he wants to do in the vicinity of the bifurcation point. Typically, a bifurcation point is detected either by a change in sign of the determinant of the stiffness matrix, or by serious convergence difficulties near the critical point. In either case, the user can detect the presence of one or more closely-spaced modes in the area of interest. NEV modes are automatically computed at run termination when NEV > 0 (ET-1) (see above). To proceed further always involves a restart.

The variable NPATH (ET-1) controls what ET does, as follows:

-1 modal analysis and computation of higher order expansion coefficients — INACTIVE

0 ordinary arc-length (Riks) path continuation algorithm.
   go to “ET-1 Solution Control” on page 11-32.

1 primary path (Equivalence Transformation ET) using the “path tangent” vector as the equivalence vector.
   go to “ET-1 Solution Control” on page 11-32.

2 branch to alternate path using single mode ET.
   go to “ET-1 Solution Control” on page 11-38.

* See, for example:
and
**Simple branch switching**

The simplest branch-switching option is a restart using $\text{NPATH} = 2$. In this case, the user wishes to branch to an alternate path in the direction of a particular mode (specified by $\text{IE}$, below). This mode is held constant, and the load factor is adjusted to make the residual vanish. The restart file must contain the mode specified by $\text{IE}$. The A-1, B-1, C-1, and D-1 records are input in the same way as an ordinary restart. Remember that load step $\text{ISTART}$ (D-1) must have been saved, along with the mode number $\text{IE}$. One can save the desired modes by setting $\text{NEV} > 0$ (ET-1) for any run with $\text{NPATH} = 0$ or 1, and $\text{INDIC} = 3, 4, \text{ or } 5$ (B-1), nonlinear analysis.

For simple mode switching, record ET-1 must be input as follows:
ET-1 Solution Control

NPATH NEV NSOL IE IGNORE LDMAX IUPLDA IUPLDB

NPATH = 2 described above
NEV ignored
NSOL
   0 – assume the solution is continuous on restart, and perform quadratic extrapolation to estimate a new solution wherever possible
   1 – assume the solution is discontinuous, and do not extrapolate to estimate the next solution. In this case, the previous solution is used as the initial estimate, and the solution will be checked for equilibrium. If equilibrium is not satisfied, stop the execution with a message; otherwise, proceed to full extrapolation after enough solutions have been accumulated.
   2 – proceed as if NSOL = 1, except attempt to obtain a new solution even if the equilibrium test fails. Users should be cautious with this option.
   3 – use the previous solution vector to start the first two iterations of each solution step; this option is particularly useful during progressive failure analysis.
IE mode number used in single-mode ET. Note that the amplitude of the mode IE is taken to be \text{STEP}(1) (D-1) times the eigenmode IE. The modes are always normalized such that their largest component is unity.
IGNORE ignored by the program; included for compatibility with old input files
LDMAX
   0 – continue solution beyond any limit points found (points of maximum load, load system A only)
   1 – stop solution upon reaching a maximum load (load system A only)
IUPLDA if nonzero, interpret the A load system as follower loads, or loads that rotate with each node
IUPLDB if nonzero, interpret the B load system as follower loads, or loads that rotate with each node
If the solution converges, s2 continues the execution using the NPATH = 1 option (see above) until either the maximum load or some other limit is exceeded.

* data deck is complete *
User-written subroutines can be used in very basic ways as elegant alternatives to lengthy input decks; or, they can be used in more sophisticated ways to extend the generality of STAGS. For example, user-written subroutines can define arbitrary functional relationships, such as spatial variation of constitutive properties in a wall fabrication, or temporal variation of loading for transient analysis. In fact, nearly all aspects of model definition, from the most basic operations to extremely complex ones, can be performed via user-written subroutines.

Important details concerning applicability of user-written subroutines, along with functional and alphabetical summaries, comprise the remainder of this introduction. Conventions for passing input data to user-written subroutines and for returning output data to STAGS are described next, in Section 12.1. A practical example problem utilizing several routines is presented in Section 12.3. In between, Section 12.2 describes each of the routines, in alphabetical order.

<table>
<thead>
<tr>
<th>Routine</th>
<th>Page</th>
<th>Routine</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CROSS</td>
<td>12-9</td>
<td>USRDGD</td>
<td>12-32</td>
</tr>
<tr>
<td>DIMP</td>
<td>12-14</td>
<td>USRELT</td>
<td>12-38</td>
</tr>
<tr>
<td>FORCET</td>
<td>12-18</td>
<td>USRFAB</td>
<td>12-40</td>
</tr>
<tr>
<td>LAME</td>
<td>12-19</td>
<td>USRFPF</td>
<td>12-32</td>
</tr>
<tr>
<td>TEMP</td>
<td>12-25</td>
<td>USRLD</td>
<td>12-55</td>
</tr>
<tr>
<td>UCONST</td>
<td>12-29</td>
<td>USRPT</td>
<td>12-57</td>
</tr>
<tr>
<td>UPRESS</td>
<td>12-30</td>
<td>WALL</td>
<td>12-59</td>
</tr>
</tbody>
</table>

**CAUTION:** Users should check documentation carefully before attempting to use any existing user-written subroutines with his or her version of STAGS.
Table 12.1  Functional summary of user-written subroutines.

<table>
<thead>
<tr>
<th>category</th>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shell Unit Geometry</td>
<td>LAME</td>
<td>Reference surface geometry</td>
</tr>
<tr>
<td>Shell Unit Initial Geometric Imperfections</td>
<td>DIMP</td>
<td>Imperfections by discrete values</td>
</tr>
<tr>
<td>Element Unit Discretization</td>
<td>USRPT</td>
<td>Nodes, coordinates and DOF definitions</td>
</tr>
<tr>
<td></td>
<td>USRELT</td>
<td>Elements connectivities and properties</td>
</tr>
<tr>
<td>Beam Cross Sections</td>
<td>CROSS</td>
<td>Beam cross section properties</td>
</tr>
<tr>
<td>Fabrications</td>
<td>USRFAB</td>
<td>Fabrication properties, via GCP</td>
</tr>
<tr>
<td></td>
<td>WALL</td>
<td>Shell wall fabrication properties</td>
</tr>
<tr>
<td>Loads</td>
<td>FORCET</td>
<td>Load factor history for transient analysis</td>
</tr>
<tr>
<td></td>
<td>TEMP</td>
<td>Thermal loadings</td>
</tr>
<tr>
<td></td>
<td>UPRESS</td>
<td>Pressure loadings</td>
</tr>
<tr>
<td></td>
<td>USRLD</td>
<td>General loadings</td>
</tr>
<tr>
<td>Displacement Constraints</td>
<td>UCONST</td>
<td>Lagrange constraints</td>
</tr>
<tr>
<td>Material Degradation and Failure</td>
<td>USRDGD</td>
<td>Material degradation (template)</td>
</tr>
<tr>
<td></td>
<td>USRFPF</td>
<td>Material failure (template)</td>
</tr>
<tr>
<td>Discontinued</td>
<td>SKEWS</td>
<td>Skew stiffeners</td>
</tr>
<tr>
<td></td>
<td>UGRID</td>
<td>Grid generation</td>
</tr>
<tr>
<td></td>
<td>WIMP</td>
<td>Imperfections by functional definition</td>
</tr>
</tbody>
</table>

Table 12.1 presents a functional summary of the user-written subroutines. Table 12.2 lists them alphabetically, summarizing the conditions under which each is applicable. Also indicated in Table 12.2 are the analysis phases for which each subroutine is needed. Note that some user-written subroutines are required in the *model definition phase*, some are required in the *primary solution phase*, and some are required in the *secondary solution phase*. Some routines are needed for more than one phase of analysis.
### Table 12.2  
Alphabetical summary, showing analysis options that indicate user-written subroutines.

<table>
<thead>
<tr>
<th>name</th>
<th>analysis option</th>
<th>location where defined</th>
<th>model</th>
<th>primary solution</th>
<th>secondary solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>CROSS</td>
<td><strong>ICROSS</strong> = 0</td>
<td>O-1a, O-2a, T-2, USRELT, K-6, WALL</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td></td>
<td><strong>ICROSM</strong> = 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIMP</td>
<td><strong>IWIMP</strong> = -1</td>
<td>M-5</td>
<td>✗</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>IUDIMP</strong> = 1</td>
<td>H-1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FORCET</td>
<td><strong>IFORCE</strong> = 1</td>
<td>E-2</td>
<td>✗</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAME</td>
<td><strong>ISHELL</strong> = 1</td>
<td>M-5</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>TEMP</td>
<td><strong>ITEMP &gt; 0</strong></td>
<td>C-1</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>UCONST</td>
<td><strong>NCONST &gt; 0</strong></td>
<td>B-2</td>
<td>✗</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UPRESS</td>
<td><strong>IPRESS</strong> = 1</td>
<td>Q-1, U-1</td>
<td>✗</td>
<td></td>
<td></td>
</tr>
<tr>
<td>USRDGD</td>
<td><strong>IDGRD = 99</strong></td>
<td>I-5a, I-10a</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>USRELT</td>
<td><strong>IUWE = 1</strong></td>
<td>H-1</td>
<td>✗</td>
<td></td>
<td></td>
</tr>
<tr>
<td>USRFAB</td>
<td><strong>FABID &lt; 0</strong></td>
<td>I-5a, T-3xx, T-4xx, USRELT</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>USRFPF</td>
<td><strong>IFAIL = 99</strong></td>
<td>I-5a, I-10a</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>USRLD</td>
<td><strong>IFLG = 1</strong></td>
<td>Q-2, U-2</td>
<td>✗</td>
<td></td>
<td></td>
</tr>
<tr>
<td>USRPT</td>
<td><strong>IUWP = 1</strong></td>
<td>H-1</td>
<td>✗</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WALL</td>
<td><strong>IWALL = 0</strong></td>
<td>M-5, T-3, T-4, USRELT</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

User-written subroutines must be linked with relevant STAGS processors to create application-specific executable code; refer to “Creating custom STAGS executables with makeuser (UNIX)” on page 2-14.

- ✔ Subroutines which are applicable in the model definition phase, such as USRPT, must be linked with the model executable, s1.
- ✔ Subroutines which are applicable in the primary solution phase, such as FORCET, must be linked with the solution executable, s2.
User-Written Subroutines

12.1 Input/Output Data Conventions

User-written subroutines are FORTRAN subroutines which receive *input* and produce *output*. This section discusses the various methods by which input data are passed to a *called* user-written subroutine from a *calling* STAGS program module and by which output data are returned to STAGS.

May user-written subroutines be coded in C?

Yes. While STAGS is a FORTRAN program, user-written subroutines may be coded in C provided that the user adheres to the conventions described in this section, and provided that the user’s programming environment supports calling C program modules from FORTRAN program modules. In fact, STAGS executables contain C code linked with FORTRAN code.
Data types

User-written subroutine input/output utilizes only two FORTRAN data types—INTEGER and REAL. The type for each data item is indicated in the corresponding description. Of course, other data types may be used for local data within each subroutine.

Input/output methods

Input/output functions in user-written subroutines utilize parameter lists, argument lists, and common blocks.

parameter lists

A parameter list is the list in a FORTRAN “subroutine” statement. For example, in the FORCET statement

```
subroutine FORCET ( t, k, Pf )
```

“( t, k, Pf )” is the parameter list. The call by reference technique is used exclusively for data in this category. For example, Pf refers to the data address where the contents of Pf are stored.

Parameter lists may be employed for both input and output functions. Not all user-written subroutines contain parameter lists.

argument lists

An argument list is the list in a FORTRAN “call” statement. For example, in the UCONST statement

```
call UCONST ( n, iu, ix, iy, id, cc )
```

“( n, iu, ix, iy, id, cc )” is the argument list. The call by reference technique generally applies here. However, call by value may be used where appropriate. For example, n may be a constant in the above list (see “UCONST Lagrange Constraints” on page 12-29). If in doubt, use a variable rather than a constant, as variables are always appropriate.

Obviously, call statements perform output only. The output data in the argument list are passed down to the called subroutine, rather than back up to the calling routine, as through parameter lists. Not all user-written subroutines output data via call statements.
**common blocks**

A FORTRAN common block contains global data, and may be employed for both input and output. For example, in subroutine USRPT

```
COMMON / NEWSY / xax, xay, xaz, yax, yay, yaz
```

is used to output the nodal-auxiliary coordinate system.

A special use of common blocks is for input of User Parameters; see “L-1 User Parameters Summary” on page 5-143. The statements

```
COMMON / UPI   / UserInt(200)
COMMON / UPP   / UserFlo(200)
```

provide access to User Parameters in any STAGS subroutine.

Not all user-written subroutines utilize common blocks.

User-written subroutines may call other user-provided subroutines. Subroutine TEMP, for example, could call a heat transfer processor to determine temperatures; or subroutine WALL could call a constitutive processor to compute constitutive properties. There are no hard limits on the allowable complexity of user-written subroutines or on the number of user-provided lower-level routines that they may call. Be careful, however, to avoid subroutine and common block name conflicts with STAGS code.
12.2 Subroutine Specifications

User-written subroutines are described in alphabetical order in this section. Some conventions that are used in the subroutine specifications are explained in the following.

User-written subroutines may be loosely grouped into three categories. Some have exact counterparts in input data records. Subroutine USRPT, for example, combines the functionalities of “S-1 User Points (upts protocol)”, “S-2 Auxiliary Coordinate System (upts protocol)”, and “U-4 Attached Mass”. Each output data item in USRPT corresponds to one from those three data records. Therefore, USRPT is used either as an alternative to, or in combination with, those input records.

Other user-written subroutines extend the generality of STAGS by providing capabilities that do not exist in the input records. For example, subroutine TEMP represents the only method for applying thermal loading in a STAGS model. Finally, there are those subroutines that combine additional capabilities with the option of specifying data just as is done on input records. For example, subroutine WALL may describe a shell wall fabrication by specifying input as in the Wall Fabrication Table (K records). In addition, WALL may define fabrication properties which vary over the surface of a shell unit, something that cannot be done with input records. Another example involves the use of subroutines USRFPP and USRDGD for users to define their own failure criteria and material degradation models.

Subroutine structure

A template is provided for each subroutine—including the necessary coding to perform I/O functions. The user need only provide the coding to perform required computations and data assignments. The source for the subroutine templates is found in the directory $STAGSHOME/examples/usersub.

Common blocks

Some subroutines utilize FORTRAN common blocks for I/O; see “common blocks” on page 12-6. In general, not all of the common blocks shown for a specific subroutine will be required for a particular application. Only those common blocks that are pertinent to the application at hand need be included in the corresponding user-written subroutine. Of course, it does no harm to include unused common data. Just be sure not to make data assignments to those items which are not applicable.
Data description

Tabular descriptions are given for all I/O data for each subroutine. Where user-written subroutine output data are identical to data on a corresponding input record, that record is referenced in the subroutine specification in lieu of including redundant documentation. For example, in subroutine USRPT (see page 12-58), the table entry

<table>
<thead>
<tr>
<th>COMMON/NEWSY/</th>
<th>S-2 (p. 7-8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>xax xay xaz</td>
<td></td>
</tr>
<tr>
<td>yax yay yaz</td>
<td></td>
</tr>
</tbody>
</table>

indicates that “category C” USRPT output data are exact counterparts of data on an S-2 input record. Referring to the S-2 record description on page 7-8, it is obvious that the implied description of category C output data are

\[
\begin{align*}
xax &= XAX \\
yax &= YAX \\
xay &= XAY \\
xaz &= XAZ \\
yay &= YAY \\
yaz &= YAZ
\end{align*}
\]

and that defining COMMON/NEWSY/ is equivalent to defining an S-2 record with the same data. This compact notation provides insight into the relationships that exist between user-written subroutines and input records.
CROSS Beam Cross Section Properties

Subroutine CROSS is called at each integration point on the reference axis of each beam having ICROSS = 0 (O-1a, O-2a, T-2, USRELT), and at each integration point on the reference surface of each shell element having a fabrication which contains smeared stiffeners having ICROSM = 0 (K-6 or WALL). CROSS enables the user to define both geometric and material cross section properties as a function of position on the reference axis or surface. This is effected by reference to the Cross Section Table and/or by definition of data directly. As an example, the basic cross section properties can be defined with a Cross Section Table ID, and eccentricity can then be defined directly as a function of position on the beam axis.

Shell-unit rings

Rings are numbered locally within each shell unit, from 1 to NRGS (F-2), in the order defined on O-1a/1b. Subroutine CROSS is called for each discrete ring having ICROSS = 0.

Shell-unit stringers

Stringers are numbered locally within each shell unit, from 1 to NSTR (F-2), in the order defined on O-2a/2b. CROSS is called for each discrete stringer having ICROSS = 0.

Element-unit beams

Elements are numbered sequentially as they are defined in each element unit, from 1 to \( n_T + n_U \), where \( n_T \) is the number of elements defined on T records and \( n_U \) is the number of elements defined in USRELT. This establishes local element numbering within each element unit. Subroutine CROSS is called for each element-unit beam element having ICROSS = 0 (defined either on a T-2 record or in USRELT).

Smeared stiffeners

Smeared stiffeners are identified by a set ID, numbered locally within each applicable shell wall fabrication from 1 to NSMRS (K-1/WALL), in the order defined on K-6 records or in WALL. Subroutine CROSS is called for each smeared stiffener having ICROSM = 0. Smeared stiffeners are applicable in both shell units and element units.
**CROSS Structure**

```fortran
subroutine CROSS (iunit, ielt, kelt, itype, XYZg, XYs, :
                xsi,   ecy,   ecz,  ilin,  iplas)

    INTEGER    maxSUB
    PARAMETER ( maxSUB = 10 )

    INTEGER      iunit,    ielt,     kelt,     itype,
                 ilin,     iplas
    REAL       XYZg(3),  XYs(2),   xsi,      ecy,      ecz

    COMMON/CROSSX/  itab,     kcross,   matb,     nsub,
                    torj,     scy,      scz,      nsoyz,
                 kapz, kapy, kapz
    INTEGER     itab,     kcross,   matb,     nsub,     nsoyz
    REAL       torj,     scy,      scz,      kapy,     kapz

    COMMON/CROSS1/  ba,       biy,      biz,      biyz,
                       soy(4),   soz(4)
    REAL        ba,       biy,      biz,      biyz,
                soy,      soz

    COMMON/CROSS2/  sa (maxSUB),   sy (maxSUB),   sz (maxSUB),
                    siy(maxSUB),   siz(maxSUB),   siyz(maxSUB),
                 isp(maxSUB)
    INTEGER    isa,       sy,       sz,
                 siy,      siz,      siyz

    COMMON/CROSS3/  y1 (maxSUB),   y2 (maxSUB),
                    z1 (maxSUB),   z2 (maxSUB),   isoc(maxSUB)
    INTEGER    isoc
    REAL       y1,       y2,       z1,       z2

    COMMON/CROSS4/  ccc(4,4),   bma
    REAL        ccc,      bma

    COMMON/CROSSM/  eb,       gb,      rhob,     alb
    REAL        eb,       gb,      rhob,     alb

      *****       Define cross section properties.       *****

      return
      end

$STAGSHOME/examples/usersub/cross.F contains the code shown in
"CROSS Structure".
```
## CROSS Data

### Input

<table>
<thead>
<tr>
<th>iunit</th>
<th>unit number</th>
</tr>
</thead>
<tbody>
<tr>
<td>ielt</td>
<td>(Refer to p. 12-9)</td>
</tr>
<tr>
<td></td>
<td>Shell-unit ring: unit local ring ID</td>
</tr>
<tr>
<td></td>
<td>Shell-unit stringer: unit local stringer ID</td>
</tr>
<tr>
<td></td>
<td>Element-unit beam: unit local element number</td>
</tr>
<tr>
<td></td>
<td>Smeared stiffener: smeared stiffener set ID</td>
</tr>
<tr>
<td>kelt</td>
<td>beam element code; e.g., 210.</td>
</tr>
<tr>
<td></td>
<td>Irrelevant for smeared stiffeners.</td>
</tr>
<tr>
<td>itype</td>
<td>beam type: 0 – element-unit beam</td>
</tr>
<tr>
<td></td>
<td>1 – shell-unit ring</td>
</tr>
<tr>
<td></td>
<td>2 – shell-unit stringer</td>
</tr>
<tr>
<td></td>
<td>3 – skew stiffener</td>
</tr>
<tr>
<td></td>
<td>4 – smeared stiffener</td>
</tr>
<tr>
<td>XYZg(3)</td>
<td>(x_g, y_g, z_g) global coordinates</td>
</tr>
<tr>
<td>XYs(2)</td>
<td>(X, Y) surface coordinates; relevant for shell units only.</td>
</tr>
</tbody>
</table>

### Output

#### A

<table>
<thead>
<tr>
<th>xsi</th>
<th>ecy</th>
<th>ecz</th>
<th>ilin</th>
<th>iplas</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ring: O-1a (p. 6-50)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>stringer: O-2a (p. 6-54)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>beam: T-2 (p. 8-9)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### B

<table>
<thead>
<tr>
<th>COMMON/CROSSX/</th>
<th>itab</th>
<th>kcross</th>
<th>matb</th>
<th>nsub</th>
<th>torj</th>
<th>scy</th>
<th>scz</th>
<th>nsoyz</th>
<th>kapy</th>
<th>kapz</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-1 (p. 5-118)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Restrictions on variation of cross-section properties

ilin, iplas (A data) and kcross (B data) must be constant within each element.

When plasticity is included (iplas ≥ 1), all data must be constant within each element.

If either of the two above restrictions is violated by CROSS (called at each integration point), STAGS will resolve the discrepancies by resetting conflicting data to the values given for integration point 1.

Cross section table selection

Set itab > 0. The remaining B data are irrelevant.
General cross section

Set itab = 0 and kcross = 1. Define B, C data.

General subelement cross section

Set itab = 0 and kcross = 2. Define B, D data.

Rectangular subelement cross section

Set itab = 0 and kcross = 3. Define B, E data.

Arbitrary cross section

Set itab = 0 and kcross = 4. Define B, F data.

Define elastic material properties

When itab = 0 only: set matb = 0 and define G data.

Note that nonlinear material properties may be defined indirectly only, by reference to a Cross Section Table entry (see “Cross section table selection” on page 12-12).
**DIMP**  Imperfections by Discrete Values

Subroutine DIMP is called for *each node point* in each shell unit with $\text{IWIMP} = -1$ on the associated M-5 record and in each element unit with $\text{IUDIMP} = 1$ on the associated H-1 record.

DIMP defines initial geometric imperfections by specifying discrete nodal displacements representing perturbations of the idealized geometry. Ordinarily, only $\textbf{(u, v, w)}$ need be defined. However, to include higher-order strain imperfections, $\textbf{(ru, rv, rw)}$ should also be defined. For shell units, $\textbf{(u, v, w)}$ are translations in the $(X', Y', Z')$ shell coordinate directions, and $\textbf{(ru, rv, rw)}$ are the corresponding rotations. This holds for all shell types, including the user-generated shell unit ($\text{ISHELL} = 1$, M-1). Figure 6.6 on page 6-58 shows the positive directions for the $\textbf{(u, v, w, ru, rv, rw)}$ components. For element units, displacements are in the $(x''', y''', z''')$ computational coordinate directions; refer to Figure 7.1 on page 7-6.

For the **E410** and **E411** shell elements, imperfections can be generated in two ways: either by perturbing the geometry alone, or by combining perturbed geometry with initial strain imperfections. The latter allows a more refined definition because the high-order cubic displacement element interpolation is made available. For most other elements, only the initial geometry is perturbed by adding the displacements from DIMP to the idealized initial geometry. It is sufficient for the great majority of applications to define only a suitable initial geometry. If $\text{IMPTHE} = 1$ (B-1, BIN file), only initial geometry is used in any case; otherwise, STAGS will attempt to raise the order of the imperfections using slope information from DIMP.

For element units, the only way to generate the higher-order information is to have access to the type of shell solid model idealization that created the model. This means access to some type of parametric surface coordinates for the surface from which the slope information can be derived. It is a safe bet that for most any model generated by an element unit, a simple perturbed initial geometry will suffice, with zero input for the rotations.

Shell units are defined analytically, according to equation (4.1) on page 4-8. Because this relationship exists, slopes can be computed, allowing rotations to be defined as

$$ru = \beta_y \quad rv = -\beta_x \quad rw = \gamma$$

Specific formulas for the so-called *bending numbers* $\beta_x$, $\beta_y$, and $\gamma$ for each STAGS shell type ($\text{ISHELL}$, M-1) are given in Appendix K. It should be noted that STAGS uses this information automatically when trigonometric expansions are input on the M-6 records.
$STAGSHOME$/examples/usersub/dimp.F contains the code shown in “DIMP Structure”.

### DIMP Structure

```fortran
subroutine DIMP (iunit, inode, prop, XYZg, XYs, U)

INTEGER
:   iunit,         inode

REAL
:   prop(8),       XYZg(3),       XYs(2),        U(6)

*****       Define U(1:6) = (u,v,w,ru,rv,rw).       *****

return
end
```

### DIMP Data

**Input**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iunit</td>
<td>unit number</td>
</tr>
<tr>
<td>inode</td>
<td>unit local node number</td>
</tr>
<tr>
<td>prop(8)</td>
<td>M-2 (p. 6-16); relevant for shell units only.</td>
</tr>
<tr>
<td>XYZg(3)</td>
<td>(x_g, y_g, z_g) global coordinates</td>
</tr>
<tr>
<td>XYs(2)</td>
<td>(X, Y) surface coordinates; relevant for shell units only.</td>
</tr>
</tbody>
</table>

**Output**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>U(6)</td>
<td>(u, v, w, ru, rv, rw)</td>
</tr>
</tbody>
</table>
Example problem: cylinder with harmonic imperfections

A cylinder with an imperfection defined as the product of an $n = 2$ (ovalizing) mode and a axial half-wave mode is described in the following example. The maximum amplitude of the imperfections is 0.01. Although this example makes rigorous use of the information in Appendix K, the user will see from the comments that many of the terms are quite small and should be omitted in a real analysis. Consult Appendix K for more details.

The example subroutine shown below is the default subroutine DIMP.

```plaintext
SUBROUTINE DIMP ( IUNIT, INODE, PROP, XYZg, XYs, VV )
C
C     DIMP requires the user to provide output in the array VV.
C     Imperfections in VV form the imperfection displacement vector
C     at a given node as follows:
C
C     VV(1) -- U
C     VV(2) -- V
C     VV(3) -- W
C     VV(4) -- 0 or Ru (beta_y)
C     VV(5) -- 0 or Rv (-beta_x)
C     VV(6) -- 0 or Rw (gamma or drilling freedom)
C
C     INPUTS
C
C     IUNIT -- Shell Unit Number
C     INODE -- Local (user's) Unit Node #
C     PROP -- PROP array from STAGS Manual
C     XYZg -- Global coordinate of node
C     XYs -- Surface coordinates
C
C     OUTPUT
C
C     VV -- Array of six displacements. Translations are in
C     units of length and rotations are in radians.
C
#include "keydefs.h"
C
INTEGER IUNIT, INODE
REAL PROP(8), XYZg(3), XYs(2), VV(6)
```
Example User Input for trigonometric representation of imperfection vector for a cylinder of length 15, radius 25.
For general DIMP, user programming replaces example below.

\[
\begin{align*}
X &= XYS(1) \\
Y &= XYS(2) \\
SPAN &= PROPS(2) - PROPS(1) \\
\pi &= 3.1415927 \\
RADIUS &= PROPS(5) \\
Y1 &= \pi\cdot Y/180. \\
WMAX &= 0.01 \\
CX &= \cos(\pi\cdot X/SPAN) \\
CY &= \cos(2.\cdot Y1) \\
SX &= \sin(\pi\cdot X/SPAN) \\
SY &= \sin(2.\cdot Y1) \\
WX &= -WMAX\cdot (\pi/SPAN)\cdot SX\cdot CY \\
WY &= -2.\cdot WMAX\cdot CX\cdot SY \\
WXY &= 2.\cdot WMAX\cdot (\pi/SPAN)\cdot SX\cdot SY
\end{align*}
\]

TRANSLATIONS (NOTE HOW SMALL VV(1) AND VV(2) ARE)

\[
\begin{align*}
VV(1) &= WX^2\cdot SPAN/2. \\
VV(2) &= WY^2/(2.\cdot RADIUS) \\
VV(3) &= WMAX\cdot CX\cdot CY
\end{align*}
\]

IN-PLANE DERIVATIVES (NOTE HOW VERY SMALL!)

\[
\begin{align*}
U_Y &= WX\cdot SPAN\cdot WXY \\
V_X &= WY\cdot WXY/RADIUS
\end{align*}
\]

ROTATIONS, FROM FORMULAS (2.24 and 2.57)

\[
\begin{align*}
BETAX &= WX \\
BETAY &= (WY-VV(2))/RADIUS \\
\gamma &= 0.5\cdot (V_X - U_Y/RADIUS) \\
VV(4) &= BETAY \\
VV(5) &= -BETAX \\
VV(6) &= \gamma
\end{align*}
\]

End Example

RETURN
END
FORCET Load Factor History for Transient Analysis

Subroutine FORCET is called when \texttt{IFORCE} = 1 on the E-2 record.

\$STAGSHOME/examples/usersub/forcet.F contains the code shown in “FORCET Structure”.

FORCET Structure

```fortran
subroutine FORCET (t, k, Pf)

INTEGER :   k
REAL :   t,       Pf

***** Define load factor. *****

return
end
```

FORCET Data

<table>
<thead>
<tr>
<th>Input</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>load set ID 1</td>
</tr>
<tr>
<td></td>
<td>load set ID 2</td>
</tr>
<tr>
<td></td>
<td>load system A</td>
</tr>
<tr>
<td></td>
<td>load system B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>load factor</th>
</tr>
</thead>
</table>
**LAME Reference Surface Geometry**

Subroutine LAME is called for *each node point* in each shell unit with ISHELL = 1 on the associated M-1 record. It specifies:

- the position of the node in \((x, y, z)\) branch coordinates given the \((X, Y)\) surface coordinates
- the tangents (also in \((x, y, z)\) branch coordinates) to the coordinate lines generated by the \((X, Y)\) surface coordinates
- a degree-of freedom (dof) option, relevant only when \((X, Y)\) are non-orthogonal

If the radius vector \(r\) from the origin of the branch (shell unit) to a given nodal point is defined by a relation of the type

\[
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix} = \begin{bmatrix}
f(X, Y) \\
g(X, Y) \\
h(X, Y)
\end{bmatrix}
\]

then the tangent vectors \(r_X\) and \(r_Y\) are

\[
\begin{align*}
\frac{\partial r}{\partial X} &= f_X \\
\frac{\partial r}{\partial Y} &= g_X \\
\frac{\partial^2 r}{\partial X^2} &= h_X \\
\frac{\partial^2 r}{\partial Y^2} &= j_X
\end{align*}
\]

The user must provide the radius vector and its partial derivatives at each node

\[
r(X, Y) \quad \frac{\partial r}{\partial X} \quad \frac{\partial r}{\partial Y}
\]

By default, PROP (M-2, p. 6-16) is used to set the range of \((X, Y)\) as

\[
PROP(1) \leq X \leq PROP(2) \quad PROP(3) \leq Y \leq PROP(4)
\]

However, when IUGRID = 1 (N-1, p. 6-34), the range of \((X, Y)\) is set to

\[
0 \leq X \leq 1 \quad 0 \leq Y \leq 1
\]

PROP remains intact, but is not used to set the range of \((X, Y)\) when IUGRID = 1; see M-2, p. 6-16, and N-1, p. 6-34.
STAGS uses the tangent vectors $\mathbf{r}_X$ and $\mathbf{r}_Y$ to generate the $(X', Y')$ shell coordinates. The $Z'$ shell normal direction is defined by the cross product $X' \times Y'$.

**Non-orthogonal surface coordinates**

When the $(X, Y)$ surface coordinates are non-orthogonal, the user selects either $\mathbf{r}_X$ or $\mathbf{r}_Y$ to determine one of $(X', Y')$. The remaining axis is chosen to complete a right-handed orthogonal system. If $\text{islam}$ is set to 1, $X'$ is in the direction defined by $\mathbf{r}_X$. If $\text{islam}$ is set to 2, $Y'$ is in the direction defined by $\mathbf{r}_Y$. For most shell definitions, the direction vectors are naturally orthogonal, and the choice of $\text{islam}$ is immaterial. If the direction vectors are not orthogonal, the choice of $\text{islam}$ may be important, since the model boundary conditions and specified displacements depend on this choice. It will always be possible to define the commonly-used boundary conditions such as symmetry and simple support with a suitable choice of $\text{islam}$.

$\texttt{STAGSHOME/examples/usersub/lame.F}$ contains the code shown in “LAME Structure”.

---

**LAME Structure**

```fortran
subroutine LAME ( iunit, prop, XYs, islam )
    COMMON / LAMEX /  f,             g,             h,
                        fX,            gX,            hX,
                        fY,            gY,            hY
    INTEGER           iunit,         islam
    REAL              prop(8),       XYs(2),
                        f              g,             h,
                        fX,            gX,            hX,
                        fY,            gY,            hY

*** Define f  g  h  fX  gX  hX  fY  gY  hY ***
***                                     ***
*** Set islam when (X,Y) surface coordinates are nonorthogonal ***

return
end
```

---
Example problem: quadrilateral plate with shell coordinates derived from (ξ, η) directions

In the Quadrilateral Plate (ISHELL = 3, M-1), the (X', Y', Z') shell coordinates are parallel to the (x, y, z) branch coordinates (see “Standard Shell Surfaces” on page 6-2). This can be inconvenient when loading and boundary conditions require that (u, v, w) on the boundary be aligned with the edge (see Figure 6.6 on page 6-58). This example solves this problem by assigning X' to the tangent line generated by the X surface coordinate (ISLAM=1) for all interior points plus boundary lines 2 and 4. For those points lying along boundary lines 1 and 3, ISLAM=2 is set, meaning that the Y surface tangent will be aligned with the Y' shell coordinate.

In this example, IUGRID = 1 (N-1, p. 6-34) must be set. This option defines the range of the (X, Y) surface coordinates as

\[ 0 \leq X \leq 1 \quad 0 \leq Y \leq 1 \]

PROP(1:8) (M-2, p. 6-16) are used to define the (x, y) branch coordinates of the four corner points according to the conventions used for the Quadrilateral Plate (ISHELL = 3, M-1).
The example subroutine shown below is the default subroutine LAME.

```fortran
subroutine LAME ( IUNIT, PROP, XYs, ISLAM )

Integer
  :     IUNIT,   ISLAM, init

save init

Real
  :     XS,   YS, PROP(*),
  :     F,   G,   H,
  :     FX,   GX,   HX,
  :     FY,   GY,   HY,   XYs(2)

#include "stndcm.h"

Common /LAMEX /
  :     F,   G,   H,
  :     FX,   GX,   HX,
  :     FY,   GY,   HY

data init/1/

C
C Given Shell Unit No. IUNIT, and surface coordinates XYs
C compute branch coordinates F, G, H of the point.
C
C Compute the First Fundamental Form, or the derivative of the
C position vector F, G, H as a function of XYs(1) (FX, GX, HX) or of
C XYs(2) (FY, GY, HY). If ISLAM = 1, STAGS aligns the local x axis
C along the vector (FX, GX, HX); if ISLAM = 2, STAGS aligns the
C local y axis along the vector (FY, GY, HY). The z axis is
C always perpendicular to both base vectors (FX, GX, HX) and
C (FY, GY, HY)
C
C In this example, create a grid for the quadrilateral plate (ISHELL=3,
C see STAGS Manual), except this time align the X' shell coordinate
C direction along the (FX, GX, HX) direction, and assign the Y' axis
C to complete a right-hand orthogonal system. Only Boundary Lines 1 and
C 3, (XS = 0. or XS = 1.), align the Y' axis with (FY, GY, HY) so that
C the local coordinates line up with the boundary lines (for loads and
C boundary conditions).
C
*** NOTE: TO RUN THIS ROUTINE, ASSIGN IUGRD = 1, Record N-1. This causes
*** 0<=XS<=1.;0<=YS<=1. Also, assign PROP array in the same manner as
*** in ISHELL = 3 (Quadrilateral Plate).
```
C
C Begin Execution
C
XS = XYs(1)
YS = XYs(2)
if (init .eq. 1) then
  init = 0
  write(not, 1)
endif
1 format(/,1x,'*************   WARNING!!   ************',/
  &       1x,'Default Lame for quadrilateral plate used.',/)
C
C Compute reduced coordinate functions
C
f1 = (1.-XS)*(1.-YS)
f2 = (1.-XS)*YS
f3 = XS*YS
f4 = XS*(1.-YS)
C
C Compute position coordinates of point at XS,YS
C
F = f1*PROP(1) + f2*PROP(3) + f3*PROP(5) + f4*PROP(7)
G = f1*PROP(2) + f2*PROP(4) + f3*PROP(6) + f4*PROP(8)
H = 0.
C
C Compute X' direction vector
C
f1x = -(1.-YS)
f2x = -YS
f3x =  YS
f4x = -f1x
FX  = f1x*PROP(1) + f2x*PROP(3) + f3x*PROP(5) + f4x*PROP(7)
GX  = f1x*PROP(2) + f2x*PROP(4) + f3x*PROP(6) + f4x*PROP(8)
HX  = 0.
C
C Compute the Y' direction vector
C
f1y = -(1.-XS)
f2y = -f1y
f3y =  XS
f4y = -XS
FY  = f1y*PROP(1) + f2y*PROP(3) + f3y*PROP(5) + f4y*PROP(7)
GY  = f1y*PROP(2) + f2y*PROP(4) + f3y*PROP(6) + f4y*PROP(8)
HY  = 0.
C     Check if on boundary line (to within real * 4 accuracy)
C
C     if (XS .lt. 1.e-5   .or. ABS(XS -1.) .lt. 1.e-5) then
C
C        You are on boundary line 1 or 3: set Y' = (FY, GY, HY)
C        by setting ISLAM = 2 (STAGS does the rest)
C
ISLAM = 2
ELSE
C
C        You are in the interior, or on lines 2 and 4: default is OK
C
ISLAM = 1
ENDIF
C
C     End computation
C
return
end
**TEMP** Thermal Loading

Subroutine TEMP is applicable if and only if $\text{ITEMP} > 0$ on the C-1 record (indicating that thermal loadings are to be taken into account during analysis and post-processing operations). When thermal loadings are taken into account, subroutine TEMP is called at each integration point on the reference surface for shell elements and for each point on the reference axis for beams. Nodal points are used instead of integration points for E48X and E8XX elements. At each reference point, TEMP is called once for each through-thickness integration point in shell wall fabrications. TEMP is also called once at the centroid of each subelement in beam cross sections (where thermal loading is permitted only for $\text{KCROSS} \leq 3$, J-1). TEMP is also called during the post-processing phase at each point selected for stress output. See the notes subsection, below, for information about thermal analysis capabilities in the current version of STAGS.

By convention, the stress-free temperature is taken as zero in STAGS. It is incumbent upon the user to define relative temperatures in TEMP: if the actual environmental condition at integration point $i$ is $T_i$ and the corresponding stress-free temperature is $\bar{T}$, the temperature defined by TEMP should be $T = T_i - \bar{T}$.

Thermal loading is added to mechanical loading as part of the base loads, which are scaled by the load factors $P_A$ and $P_B$ (see Section 6.5 “Loads” on page 6-64). Thermal loading is added exclusively to either Load System A or Load System B, as specified by $\text{ITEMP}$, C-1.

`$\text{STAGSHOME}/examples/usersub/temp.F` contains the code shown in “TEMP Structure”.

**TEMP Structure**

```fortran
 subroutine TEMP ( iunit, ielt, kelt, itype,
                   XYZg,  XYs,  Zfab, YZsec, h, T )
  INTEGER  iunit,   ielt,   kelt, itype
  REAL     XYZg(3), XYs(2), Zfab, YZsec, h, T

  ***** Define temperature, T. *****
  return
  end
```

STAGS 5.0 User Manual
April, 2009

12-25
**TEMP Data**

### Input

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iunit</td>
<td>unit number</td>
</tr>
<tr>
<td>ielt</td>
<td>element number (unit local element ID)</td>
</tr>
<tr>
<td>kelt</td>
<td>element code; e.g., 410</td>
</tr>
<tr>
<td>itype</td>
<td>element type: 1 – shell element 2 – element-unit beam 3 – shell-unit ring 4 – shell-unit stringer</td>
</tr>
<tr>
<td>XYZg(3)</td>
<td>(x_g, y_g, z_g) global coordinates</td>
</tr>
<tr>
<td>XYs(2)</td>
<td>(X, Y) surface coordinates</td>
</tr>
<tr>
<td>Zfab</td>
<td>ẑ, fabrication normal coordinate</td>
</tr>
<tr>
<td>YZsec(2)</td>
<td>(y, z) cross-section coordinates</td>
</tr>
<tr>
<td>h</td>
<td>shell wall thickness</td>
</tr>
</tbody>
</table>

- **XYs** is relevant for *shell units* only.
- **Zfab** and **h** are relevant for *shell elements* only.
- **YZsec** is relevant for *beam elements* only.
- The remaining data are always relevant.

### Output

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>temperature</td>
</tr>
</tbody>
</table>
Example problem: linear through-thickness shell-wall temperature gradient

A linear thermal gradient through the thickness of a shell wall may be expressed as

\[ T_0 = \frac{(T_{\text{top}} + T_{\text{bot}})}{2} \quad \Delta T = T_{\text{top}} - T_{\text{bot}} \quad T = T_0 + \frac{\Delta T}{h} z \]

where \( z \) is the fabrication normal coordinate, \( h \) is the wall thickness, \( T_0 \) is the temperature at \( z = 0 \) (i.e., the middle-surface temperature), \( T_{\text{top}} \) is the top-surface temperature, and \( T_{\text{bot}} \) is the bottom-surface temperature (see Figure 5.6 on page 5-130).

Utilizing User Parameters (see “L-1 User Parameters Summary” on page 5-143) to input the top-surface and bottom-surface temperatures,

\[ T_{\text{top}} = T_{\text{top}} \quad \text{and} \quad T_{\text{bot}} = T_{\text{bot}} \]

TEMP can define a linear temperature variation as in the code shown below.

```fortran
subroutine TEMP (iunit, ielt, kelt, itype,
:                 XYZg, XYs, Zfab, YZsec, h, T)
COMMON
: / UPF   /   Ttop,     Tbot,     UserFlo(198)
INTEGER
:   iunit,         ielt,          kelt,          itype
REAL
:   XYZg(3),       XYs(2),        Zfab,          YZsec(2),
:   h,             T,             Ttop,          Tbot,
:   Tmid,          delT,          Tgrad
Tmid  = (Ttop + Tbot)/2
delT  =  Ttop - Tbot
Tgrad = delT/h
T = Tmid + Tgrad*Zfab
return
end
```

The example subroutine shown above is the default subroutine TEMP.
Notes:

In a recent effort to evaluate thermal analysis capabilities in the current version of STAGS for shell and solid elements and in doing so to correct “simple” errors that were found, the following observations about STAGS’ thermal analysis capabilities and shortcomings were reported:

Test cases for free thermal expansion provide excellent indications that the machinery in STAGS for thermal loading is working. Standard free thermal expansion analysis cases were run—using “historical” material and fabrication specification options and using GCP (Generalized Constitutive Processor) material and fabrication specification options—for all of the basic shell and solid elements in the current version of STAGS. For shell elements, the default plane-stress material response should yield zero energy for free expansion. All of the shell and solid elements in the current version of STAGS passed this test.

The E330, E430 and E480 shell elements must be defined with GCP material and fabrication specifications when thermal loadings are applied: these elements do not take thermal loadings into account when the historical material- and fabrication-specification methods are used. To obtain correct energy printouts with these elements, three or more integration points must be used through the thickness of each layer. This is needed for accuracy anyway, when thermal gradients are present. A single integration point may be used in each layer of a fabrication with many layers—if the analyst’s user-written TEMP subroutine takes that into account.

With the E320, E410 and E420 shell elements in the current version of STAGS, the plane-strain material option is only available for ISOTROPIC materials that are specified via the “historical” procedures. This option is not implemented via the GCP, yet. Results obtained for these elements (with historical plane-strain specifications) were identical with results obtained with STAGS’ solid elements (with responses in the thickness direction suppressed).

To implement a plane-strain option in the GCP, a new kinematic class must be added. The new kinematic class (which can be implemented with a few weeks of funded effort) is essential for the E330, E430 and E480 shell elements. It is not critically required for STAGS’ E8XX sandwich elements, though: plane strain behavior can be modeled via boundary conditions, with them.

There is not enough information in the “historical” material and fabrication specification procedures in the current version of STAGS to specify generalized plane-strain materials: material properties in the thickness direction are not defined. That information could be added to the historical procedures, but the cost/benefit figures for doing so are unattractive (at best). Much better results are anticipated from enhancements to the GCP.
**UCONST** Lagrange Constraints

Subroutine UCONST is called when `NCONST > 0` on the B-2 record.

**UCONST Structure**

```fortran
subroutine UCONST
    INTEGER     maxN
    PARAMETER ( maxN = 100 )
    INTEGER  iu(maxN), ix(maxN), iy(maxN), iz(maxN), id(maxN)
    REAL     cc(maxN)
    INTEGER  nterms

    ***** Repeat for each constraint. *****
    call CONSTR ( nterms, iu, ix, iy, iz, id, cc )
    return
end
```

**UCONST Data**

**Output**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>nterms</td>
<td>G-3 (p. 5-40)</td>
<td></td>
</tr>
<tr>
<td>iu  ix iy iz id cc</td>
<td>G-4 (p. 5-43)</td>
<td></td>
</tr>
</tbody>
</table>

$STAGSHOME$examples/usersub/uconst.F contains the code shown in “UCONST Structure”. See Section 12.3 “Example Problem” on page 12-66 for a practical example of UCONST.
UPRESS Pressure Loading

Subroutine UPRESS is called at each integration point on the reference surface of each shell element where \( \text{IPRESS} = 1 \) on the associated Q-1 (shell unit) or U-1 (element unit) record. (Nodal points are used instead of integration points for elements E480 and E6XX.)

Applied loads are normally computed as the product of base loads and load factors, summed for load systems A and B (see Section 6.5 “Loads” on page 6-64). UPRESS provides a more generalized pressure-loading feature. Input arguments include time (\( t \)) and load factors (\( p_A, p_B \)), and output is a pressure value (\( p \)) which is added directly to the applied loads (i.e., UPRESS output is not placed in either load system and is not scaled). Normally, UPRESS should define the pressure, \( p \), as

\[
p = p_A \cdot p_A + p_B \cdot p_B
\]

where \( p_A, p_B \) are the load factors for load systems A and B; and \( p_A, p_B \) are the base load pressure values for the two load systems. UPRESS permits more general pressure loading, however.

Subroutine UPRESS is called at each load/time step, and may allow pressure distribution (base loading) to vary during the analysis history in any arbitrary way. UPRESS is the only means by which base loads may be varied during an analysis. For all other loading features, only load factors vary, with base loads held constant.

When subroutine UPRESS is utilized, any pressure loads defined in the INP file (\( LT = 4 \) or 5, Q-3/U-3) are ignored.

$STAGSHOME/examples/usersub/upress.F contains the code shown below:

**UPRESS Structure**

```fortran
subroutine UPRESS ( t, pA, pB, iunit, ielt, kelt,
                   :                      XYZg, XYs, live, press )

INTEGER iunit, ielt, kelt, live
REAL    t, pA, pB, XYZg(3), XYs(2), press

***** Define pressure. *****

return
end
```
## UPRESS Data

### Input

<table>
<thead>
<tr>
<th>t</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>pA</td>
<td>load factor for load system A</td>
</tr>
<tr>
<td>pB</td>
<td>load factor for load system B</td>
</tr>
<tr>
<td>iunit</td>
<td>unit number</td>
</tr>
<tr>
<td>ielt</td>
<td>element number (unit local element ID)</td>
</tr>
<tr>
<td>kelt</td>
<td>element code; e.g., 410</td>
</tr>
<tr>
<td>XYZg(3)</td>
<td>(x_g, y_g, z_g) global coordinates</td>
</tr>
<tr>
<td>XYs(2)</td>
<td>(X, Y) surface coordinates</td>
</tr>
</tbody>
</table>

- ✔ XYs is relevant for shell units only.
- ✔ The remaining data are relevant for shell units and element units.

### Output

<table>
<thead>
<tr>
<th>live</th>
<th>pressure type 0 – dead pressure 1 – live pressure</th>
</tr>
</thead>
<tbody>
<tr>
<td>press</td>
<td>pressure value</td>
</tr>
</tbody>
</table>
USRDGD Material Degradation Model

Subroutine USRDGD allows the STAGS user to implement his or her own material degradation model or to modify one of the existing degradation models (which are implemented in the current version of STAGS in DEGRADi.F subroutines). The existing first-ply failure models (which are implemented in STAGS in FPfi.F routines) can be utilized with the user’s USRDGD subroutine if they fit the user’s model. It will be necessary for the user to write his or her own USRFPPF subroutine to implement a new failure model if they do not fit. Subroutine USRDGD is called [by STAGS’ Generic Constitutive Processor (GCP) facility] for each element that is fabricated with an orthotropic elastic brittle GCP material for which IDGRD = 99 on the l-10a (ORT_EL_BR_MATERIAL) record that defines that material.

As a template for adventuresome STAGS users, a material degradation model using the maximum strain criteria has been implemented in the USRDGD subroutine that is distributed with the program (and which is listed below for the reader’s convenience). The calling arguments to USRDGD are documented in this listing and will not be further described here.

USRDGD Template

c=deck    usrdgd
c=purpose Material degradation model using a user-written routine
c=author  Norm Knight/Veridian
c=version January 2000

#include "keydefs.h"

*  *********************************************************************
*  *  This routine is provided as a template for a user that wants to  *
*  *  install a personal material degradation model. It is called iff  *
*  *  the IDGRD parameter in the ORT_EL_BR_MATERIAL GCP material       *
*  *  *  model data is set equal to 99                                    *
*  *********************************************************************

#if   _usage_
*     ------------------------------------------------------------------
*     CALLING SEQUENCE:
*     call USRDGD ( iCMtyp, mpd,   oldhmd, newhmd,
*       &              nwold,  nwnew, flag,   status )
*     *
*     INPUT ARGUMENT TYPE DESCRIPTION
*     *  icMtyp    [I]   Kinematic option for material processing
*     *  mpd(*)   [R]    Material property data
*     *  nwold    [I]   Number of words of old historical data
*     *  oldhmd(*) [R]   Historical data (from previous converged soln)
*     *  flag(*)  [I]   Failure flag vector
*     *
*     OUTPUT ARGUMENTS:
*     *  nwnew    [I]   Number of words of new historical data
*     *  newhmd(*) [R]  New historical data

12-32 April, 2009 STAGS 5.0 User Manual
* status         [I]  Return status: (OK==>status=qOK)
*     ---------------------------------------------------------------
#endif

subroutine USRDGD ( iCMtyp, mpd, oldhmd, newhmd,
                     nwold, nwnew, flag, status )

    _implicit_none_
#include "cs4xxx.h"
#include "qgcp.h"
#include "stndcm.h"
#include "upfi.h"

    integer iCMtyp
    real    mpd(*)
    _float_ oldhmd(*)
    _float_ newhmd(*)
    integer nwold
    integer nwnew
    integer flag(*)
    integer status

* The original input material data are stored in the mpd array
* in the order given in the ORT_EL_BR_MATERIAL material data
* input line for the GCP data

* INTERNAL DECLARATIONS
*     ---------------------------------------------------------------

    integer numcol
    _float_ E1, E2, E3
    _float_ G23, G13, G12
    _float_ nu23, nu13, nu12

* dfactr is equivalent to beta in the write-ups and
* it is used to degrade the material properties
    _float_ dfactr
* ... BETA=dfactr ... *
*     If beta (which is mpd(39)) is less than zero, then degrade by
*     beta only once on initial failure; if beta is positive, then
*     degrade recursively on each solution step
    dfactr = ABS(mpd(39))

* LOGIC
*     ---------------------------------------------------------------

* Set up nwnew and the newhmd array with new
* data based on element type
    if ( iCMtyp.eq.1 ) then
*  1D CONTINUUM (ROD OR BAR ELEMENT)
*     =================================
    numcol = 1
    elseif ( iCMtyp.eq.2 ) then
*  C0/C1 BEAM
*     =========
    numcol = 3
elseif ( iCMtyp.eq.3 ) then

* C0 SHELL
* ========
numcol = 5

elseif ( iCMtyp.eq.4 ) then

* C1 SHELL (PLANE STRESS)
* ================
numcol = 3

* First load in constitutive data into local variables
* using original data if nwold=0 and historical data
* if nwold .ne. 0
if ( nwold.eq.0 ) then

* Set original material data values in
* elastic constant variables
E1   = mpd(1)
E2   = mpd(2)
E3   = mpd(3)
G23  = mpd(4)
G13  = mpd(5)
G12  = mpd(6)
u23  = mpd(7)
u13  = mpd(8)
u12  = mpd(9)
else

* Set saved historical data values in elastic constant
* variables (values from the last previously converged
* and archived set)
E1   = oldhmd(1)
E2   = oldhmd(2)
E3   = oldhmd(3)
G23  = oldhmd(4)
G13  = oldhmd(5)
G12  = oldhmd(6)
u23  = oldhmd(7)
u13  = oldhmd(8)
u12  = oldhmd(9)
endif

* Degrade properties if failure criterion indicates failure
if ( flag(1) .ne. 0 ) then

* Fiber failure
if ( mpd(39).lt.0.0 .and. NINT(oldhmd(10)).ne.0 ) then
   E1   = oldhmd(1)
u12  = oldhmd(9)
else
   E1   = dfactr*E1
   nu12 = dfactr*nu12
endif
endif
if ( flag(2).ne.0 ) then

* Matrix failure
if ( mpd(39).lt.0.0 .and. NINT(oldhmd(11)).ne.0 ) then
  E2   = oldhmd(2)
  nu12 = oldhmd(9)
else
  E2   = dfactr*E2
  nu12 = dfactr*nu12
endif
endif
if ( flag(3).ne.0 ) then
  * In-plane shear failure
  if (mpd(39).lt.0.0 .and. NINT(oldhmd(12)).ne.0 ) then
    G12 = oldhmd(6)
  else
    G12 = dfactr*G12
  endif
endif

* Set nwnew = number of words of constitutive historical data
nwnew = 12
* Store new properties for this point
newhmd( 1) = E1
newhmd( 2) = E2
newhmd( 3) = E3
newhmd( 4) = G23
newhmd( 5) = G13
newhmd( 6) = G12
newhmd( 7) = nu23
newhmd( 8) = nu13
newhmd( 9) = nu12
newhmd(10) = flag(1)
newhmd(11) = flag(2)
newhmd(12) = flag(3)
elseif ( iCMtyp.eq.5 ) then
  * 2D PLANE STRAIN CONTINUUM
  numcol = 3
elseif ( iCMtyp.eq.6 ) then
  * 2D axisymmetric continuum
  numcol = 4
elseif ( iCMtyp.eq.7 ) then
  * 3D CONTINUUM
  numcol = 6
endif

* SET STATUS VARIABLE
* status = qOK
end
The user may include data in addition to that which is transmitted to and from subroutine USRDGD through its calling sequence via STAGS' user parameter input facility—using L-2a and/or L-2b records as required (for integers and real variables, respectively). This may be accomplished by including the upfi.h header file or by including the following explicit declarations and common statements:

```plaintext
integer userint
common /upi/ userint(200)
and
real userflo
common /upf/ userflo(200)
```

The csrxxx.h header file that is included in this version of USRDGD contains four integer variables that are used to identify which material point is being examined:

- **idelt**: Element number within a unit
- **eltip**: Element surface integration point number
- **layer**: Layer number within the laminate
- **layip**: Through-the-thickness integration point within a layer

On entry into the routine, a check is made to identify the kinematics option for the element by testing the value of the parameter *iCMtyp*. For C¹ shell elements, *iCMtyp* equals 3, and there are three stress values ($\sigma_{xx}$, $\sigma_{yy}$, $\tau_{xy}$). Similarly, there are three strain values.

Next, the value of *nwold* is checked. If *nwold* is zero, then no failure at this material point has occurred previously (*i.e.*, at a previous solution step, not at a previous iteration); and the original values for the elastic constants from the *mpd* array are loaded into local variables for these material constants. For convenience, local variables for a three-dimensional elastic system are stored. If *nwold* is nonzero, however, then failure at this material point has occurred previously; and the degraded elastic constants from the previous solution step are loaded from the *oldhmd* array into the local variables.

The next step is to test to determine if failure in the fiber (or 1) direction has occurred during this step (if *flag*(1) is not zero) or if no failure has yet occurred (*flag*(1) equals zero). If *flag*(1) is not zero, then the material properties are degraded in a manner that depends on whether the failure is tensile (*flag*(1) = +1) or compressive (*flag*(1) = -1). Before the properties are degraded, the type of degradation to use is determined based on the sign of *mpd*(37) that corresponds to the variable $\beta$ in the ORT_EL_BR_MATERIAL input for GCP. If this value is negative, then the degradation factor is applied only once. If it is positive, then the degradation factor is applied recursively for each solution step. As such, the properties may be degraded instantaneously to nearly zero, or they may be gradually reduced over several solution steps. If *flag*(1) equals zero, then no material degradation is done. A check is also made to determine if failure occurred during the previous solution step so that an initial degradation of the material properties is...
performed. This test examines the entries in the \texttt{oldhmd} array corresponding to the failure flags (e.g., \texttt{oldhmd}(10) corresponds to \texttt{flag}(1) and so on). The process continues with \texttt{flag}(2) and \texttt{flag}(3) in a similar manner.

After testing each value of the \texttt{flag} array, the new values of the elastic constants are stored in the \texttt{newhmd} array. Again for convenience, the elastic constants for a three-dimensional solid are stored. In addition, the three failure flags are also stored. Hence a total of \texttt{nnew} real numbers are stored in the \texttt{newhmd} array for each material point that has failed. In this template, \texttt{nnew} equals 12 because there are nine elastic constants and three failure flags.

Subroutine \texttt{USRDGD} is called for each layer integration point through the thickness of each layer in the laminate for each surface integration point of the element and for each element. It is a very low-level routine that is called many, many times; and if the user should choose to print output within this routine, he or she should expect a \textit{lot}!

Because \texttt{USRDGD} is a low-level routine, it is called during the evaluation of both the first and second variations during a nonlinear iteration. Hence multiple passes through this routine may occur as a consistent state for the first and second variation is always sought.


**USRELT Elements**

Subroutine USRELT is called once for each element unit where $\text{IUWE} = 1$ on the associated H-1 record. USRELT may define all of the elements in a given element unit, or it may be used to complement elements defined on T records. Elements are numbered sequentially as they are defined in USRELT, from $n_T + 1$ to $n_T + n_U$, where $n_T$ is the number of elements defined on T records, and $n_U$ is the number of elements defined in USRELT. Thus, element numbering is continuous, from 1 to $n_T + n_U$ within each element unit.

**USRELT Structure**

```fortran
subroutine USRELT ( iunit )

INTEGER maxN
PARAMETER ( maxN = 9 )
INTEGER :	 iunit,
: node(maxN), kelt, icross, iwall,
: ilin, iplas, integ, ipenl,
: iang
REAL :	 xsi, zeta, ecy, ecz,
: rx, ry, rz

***** Repeat for each beam element.  *****
  call BEAM (node, kelt, icross, 
: xsi, ecy, ecz, 
: ilin, iplas)

***** Repeat for each shell element.  *****
  call SHELL (node, kelt, iwall, 
: zeta, ecz, 
: ilin, iplas, 
: integ, ipenl, iang, 
: rx, ry, rz )

return
end
```
## USRELT Data

### Input

<table>
<thead>
<tr>
<th>iunit</th>
<th>unit number</th>
</tr>
</thead>
</table>

### Output

<table>
<thead>
<tr>
<th>node</th>
<th>kelt</th>
<th>icross</th>
<th>xsi</th>
<th>ecy</th>
<th>ecz</th>
<th>ilin</th>
<th>iplas</th>
<th>T-2 (p. 8-9)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>See below re: node array.</td>
</tr>
</tbody>
</table>

| node | kelt | iwall | zeta | ecz | ilin | iplas | integ | ipenl | iang | Triangle: T-3 (p. 8-14) |
|------|------|-------|------|-----|------|-------|-------|-------|------| Quadrilateral: T-4 (p. 8-17) |
| B    |      |       |      |     |      |       |       |       |      | T-4a (p. 8-23) |
|      |      |       |      |     |      |       |       |       |      | See below re: node array. |

| rx   | ry   | rz   | Triangle: T-3a (p. 8-16) |
| C    |      |      | Quadrilateral: T-4b (p. 8-24) |

✔ **A** data applies to beams.

✔ When icross = 0, then the remaining **A** data are irrelevant.

✔ **B, C** data apply to shells.

✔ When iwall = 0, then the remaining **B** data, except iang, is irrelevant.

✔ **C** data are relevant only when iang = 1.

✔ integ and ipenl are not used for triangles, but must appear in the call SHELL argument list.

### Beams

- node(1) = N1
- node(2) = N2
- node(3) = NR

### Triangles

- node(1) = N1
- node(2) = N2
- node(3) = N3

### Quadrilaterals

- node(1) = N1
- ... node(9) = N9

$STAGSHOME/examples/usersub/usrelt.F$ contains the code shown in “USRELT Structure”. See Section 12.3 “Example Problem” on page 12-66 for a practical example of USRELT.
USRFAB User-Specified Material & Fabrication Properties

User-written subroutine USRFAB, like subroutine WALL, enables the user to specify material and/or fabrication properties at each surface integration point of selected elements. Subroutine WALL lets the user do this for shell elements for which material and fabrication properties are defined via historical methods (via I*, J* and K* records, but not with the GCP), when it is necessary or desirable to circumvent or supplement STAGS’ regular property specification procedures. The user does this with subroutine WALL by setting the IWALL parameter equal to zero for any shell–and/or element–unit element(s) for which some or all of the material and/or geometric properties must be specified individually (see M-5, T-3, T-3xx, T-4xx and USRELT, for example). Subroutine USRFAB enables the user to specify material and/or fabrication properties at each surface integration point, for each of the layer points through the thickness, of selected elements that utilize the GCP for specification of and operations with these properties.

When a user-written GCP fabrication—which requires the specification of a GCP material and the specification of the GCP fabrication (geometric and control) parameters to be used—is desired for selected elements, the user (in his or her model-specification input file for STAGS’ s1 processor) must specify nominal (default) GCP material properties for these elements in the usual way (via I-6a, I-7a, I-8a, I-9a, I-10a and/or I-11a material-property-specification records, as appropriate); and he (or she) must specify nominal (default) GCP fabrication properties via l-21a and/or l-22a fabrication-specification records that are commanded by l-5a records on which the fabrication-identification parameter INFO(1) (which translates to FABID on l-21a and l-22a) is negative. When an element that has been assigned this “tagged” fabrication is encountered in STAGS’ s2 processor (or in other post-model-generation STAGS processors), the original material and fabrication properties are loaded into their appropriate header files (common blocks). User-written subroutine USRFAB is then called for each integration point—to redefine zero or more of these properties, as and if it is necessary to do so.

The user’s USRFAB subroutine must not change the number of layers in the fabrication; but it can modify most of the other fabrication properties (thicknesses, orientation angles, etc.), as and if it is necessary to do so.

The user’s USRFAB subroutine must not change the material type (ISOELASTIC, ORTHOELAST, PLASTIC_WB, etc.); but it can select and use a different material of the same type, if desired. On entry into subroutine USRFAB, all of the data for the original (nominal) material are defined in the appropriate header file(s) for that material (see the header-file listings following the USRFAB Structure table). If any or all of these data are modified for any given layer(s), USRFAB must call STAGS’ subroutine MATSET (to identify the layer(s) that have changed) before returning to the calling program. For a layered shell fabrication with a material that varies from layer to layer, USRFAB must specify changes (from the nominal material properties) for one or more layers, then call MATSET to identify the modified layer(s). This process must be repeated until all
modified layers have been changed and identified. Each call to MATSET generates a material data entry that will be used in various routines to load material data needed for computations.

The FORTRAN code shown in the “USRFAB Template” listing is available in the $STAGSHOME/examples/usersub directory. The user must replace the boldfaced “write...,” “format...” and “STOP” lines, on page 12-44, with coding that is appropriate for his or her problem. The header files referenced in the boldfaced include statements, on page 12-44, are shown after this “USRFAB Template” listing. The calling sequence for subroutine MATSET is documented immediately after these header file listings.
USRFAB Template

c=deck usrfab
c=purpose Template for user-written subroutine USRFAB
c=version May, 2002

#include "keydefs.h"

#if _usage_
 *
 * Calling sequence:
 *
 * call USRFAB ( t, Pa, Pb, iunit,
 *     ielt, kelt, kfab, eltip,
 *     XYZg, XYs, ntvals, tvals,
 *     nlayrs, lays, laymat, laythk,
 *     layint, layang, zeta, ecz,
 *     ilin, iplas )
 *
 *
 Input Arguments
 * ============
 * t = Time (seconds)
 * Pa = Load factor for system A
 * Pb = Load factor for system B
 * iunit = Unit number; unit = 0 specifies the entire model
 * ielt = Local element number within the specified unit; when
 *     unit = 0, elt specifies the global elt number
 * kelt = 1 -- Unit is a shell unit
 *       = 2 -- Unit is an element unit
 * kfab = Fabrication number assigned for this element
 * eltip = Surface (volume) integration point number in element
 * XYZg = Global coordinates at integration point
 * XYs = Shell X,Y coordinates at integration point
 * ntvals = Number of temperature sampling points
 * tvals = Temperature gradient at sampling points
 * nlayrs = Number of layers in fabrication KFAB
 * lays = Integer array for (optional) use in call to MATSET
 *
 Output Arguments
 * ===========
 * laymat(j) = Material identifier for layer j
 * layint(j) = # of through-layer integration pts for layer j
 * laythk(j) = Thickness of layer j
 * layang(j) = Fabrication orientation angle of layer j
 * zeta = Angle from wall-ref coord to fabrication coord
 * ecz = Eccentricity in Z' dirn (Z' coord of mid surface)
USRFAB Template (continued)

* ilin = 0 -- Nonlinear strain-displacement relations
* = 1 -- Linear strain-displacement relations
* iplas = 0 -- Elastic material properties used
* = 1 -- Plasticity theory enforced at all integ pts
* = 2 -- Plasticity theory enforced at elt centroid
*
#endif

*******************************************************************
subroutine USRFAB ( t, Pa, Pb, iunit,
                   & ielt, kelt, kfab, eltip,
                   & XYZg, XYs, ntvals, tvals,
                   & nlayrs, lays, laymat, laythk,
                   & layint, layang, zeta, ecz,
                   & ilin, iplas )
*******************************************************************

_implicit_none_

Real t
Real Pa
Real Pb
Integer iunit
Integer ielt
Integer kelt
Integer kfab
Integer eltip
Real XYZg(3)
Real XYs(2)
Integer nlayrs
Integer ntvals
Real tvals(ntvals)
Integer lays(nlayrs)
Integer laymat(nlayrs)
Real laythk(nlayrs)
Integer layint(nlayrs)
Real layang(nlayrs)
Real zeta
Real ecz
Integer ilin
Integer iplas
USRFAB Template (continued)

```c
#include "mater1.h"
#include "mater2.h"
#include "mater3.h"
#include "mater4.h"
#include "mater7.h"
#include "mater8.h"
#include "mater9.h"
#include "mater10.h"
#include "stndcm.h"

Logical debug
Logical NTITLE

* ================
* MATERIAL TYPE CODES:
* ================
*
* Code  Items   Description
* ----  -----   ------------
*  1     7    Linear elastic isotropic material
*  2    18    Linear elastic orthotropic material
*  3    54    Mechanical sub-layer plasticity material
*  4    44    Linear elastic orthotropic brittle material
*  5    12    Shape-memory-alloy material
*  6    54    Plane-strain material
*  7    36    PDCOMP/PDLAM property material
*  8    32    Abaqus umat material
*  9    19    SHM membrane material
* 10    10    Hahn nonlinear elastic orthotropic material

download = .false.
if (NTITLE('X_UsrFab')) debug = .true.

write (not,1000)
1000 format ('ERROR: Subroutine USRFAB has not been provided.' )
STOP
end
```
**mater1.h Header File**

* *************************************************************
* mater1.h header file,                                       *
* for Linear elastic isotropic material                       *
* *****************************************************************

Real            e_1, gnu_1, rho_1, alpha_1, beta_1, t_1, m_1
common /mater1/ e_1, gnu_1, rho_1, alpha_1, beta_1, t_1, m_1

**mater2.h Header File**

* *************************************************************
* mater2.h header file,                                       *
* for Linear elastic orthotropic material                     *
* *****************************************************************

Real            e1_2,  e2_2,  e3_2,  g12_2, g13_2, g23_2
Real            p12_2, p13_2, p23_2, rho_2, a1_2,  a2_2, a3_2
Real            b1_2,  b2_2,  b3_2,  T_2,   M_2
common /mater2/ e1_2,  e2_2,  e3_2,  g12_2, g13_2, g23_2,
&             p12_2, p13_2, p23_2, rho_2, a1_2,  a2_2, a3_2,
&             b1_2,  b2_2,  b3_2,  T_2,   M_2

**mater3.h Header File**

* *************************************************************
* mater3.h header file,                                       *
* for mechanical sublayer plasticity material                *
* *****************************************************************

Real            e_3, gnu_3, rho_3, alpha_3
Integer         nsubs
Real            ee_ss_3
common /mater3/ e_3, gnu_3, rho_3, alpha_3,
&             nsubs,
&             ee_ss_3(2,20)
**mater4.h Header File**

* *********************************************************
* mater4.h header file,
* for linear orthotropic elastic brittle matl
* *********************************************************

Real            E1_4,    E2_4,    E3_4
Real            G12_4,   G13_4,   G23_4
Real            P12_4,   P13_4,   P23_4,    Rho_4
Real            a1_4,    a2_4,    a3_4
Real            b1_4,    b2_4,    b3_4,     T_4,     M_4
Real            e1c_4,   e1t_4,   e2c_4,   e2t_4,   gam12f_4
Real            e3c_4,   e3t_4,   gam23f_4, gam13f_4
Real            Xc_4,    Xt_4,    Yc_4,     Yt_4,    Sxy_4
Real            Zc_4,    Zt_4,    Syz_4,    Sxz_4
Real            alpha_4, F12_4,   beta_4,   IPAIL_4, IDGRD_4
Real            visf0_4, visf1_4, visff_4

common /mater4/ E1_4,    E2_4,    E3_4,
&                G12_4,   G13_4,   G23_4,
&                P12_4,   P13_4,   P23_4,    Rho_4,
&                a1_4,    a2_4,    a3_4,
&                b1_4,    b2_4,    b3_4,     T_4,     M_4,
&                e1c_4,   e1t_4,   e2c_4,   e2t_4,   gam12f_4,
&                e3c_4,   e3t_4,   gam23f_4, gam13f_4,
&                Xc_4,    Xt_4,    Yc_4,     Yt_4,    Sxy_4,
&                Zc_4,    Zt_4,    Syz_4,    Sxz_4,
&                alpha_4, F12_4,   beta_4,   IPAIL_4, IDGRD_4,
&                visf0_4, visf1_4, visff_4
USRFAB must call STAGS’ subroutine MATSET if any material properties are changed. The three arguments in the calling sequence for MATSET are documented in the following listing. Note: MATSET is a STAGS routine that the user should not modify under any conditions.

c=deck matset
c=purpose STAGS subroutine that is called from user-written USRFAB
c=purpose to set modified material data for a group of layers at a
c=purpose single surface integration point of a given element
c=version May 2002

#include <stdio.h>

void MATSET(int imatl, int nlayrs, int lays[100]){
    // Implementation details...
}

The following example shows the “business” part of a typical user-written USRFAB subroutine (i.e., the coding that replaces the three boldfaced lines at the bottom of the USRFAB template that is shown on page 12-44):

```c
mater8.h Header File

*  ******************************************************
*  mater8.h header file, for Abaqus UMAT material
*  ******************************************************

Real E1_8, E2_8, P12_8, G12_8
Real XT_8, YC_8, XC_8, ALPHA_8
Real NLAY_8, LSF_8, NOTOEL_8, MONINODE_8
Real FIVISC_8, DTIME_8, DFMIN_8, DPHIO_8
Real DELTA_8, BETA_8_8, ETA_8_8, BETAC_8
Real IORX_8, SDPARA_8, VRUINC_8, VISFAC_8
Real NIFVE_8, DAPP_8, IDAMP_8, ZR1_8
Real ZR2_8, G23_8_8, G13_8_8, RHO_8

common /mater8/ E1_8, E2_8, P12_8, G12_8,
                  XT_8, YC_8, XC_8, ALPHA_8,
                  NLAY_8, LSF_8, NOTOEL_8, MONINODE_8,
                  FIVISC_8, DTIME_8, DFMIN_8, DPHIO_8,
                  DELTA_8, BETA_8_8, ETA_8_8, BETAC_8,
                  IORX_8, SDPARA_8, VRUINC_8, VISFAC_8,
                  NIFVE_8, DAPP_8, IDAMP_8, ZR1_8,
                  ZR2_8, G23_8_8, G13_8_8, RHO_8
```
Integer i
Integer mattyp
Real e1_s1 (2,3)

data (e1_s1(1,i),i=1,3) / 0.014,
&                          0.029,
&                          0.060 /
data (e1_s1(2,i),i=1,3) / 3628.39988,
&                          6200.0,
&                          8600.0 /

if ( iunit.eq.7 ) then
  if ( (ielt.ge.37 .and. ielt.le.40) .or.
    &        (ielt.ge.46 .and. ielt.le.49) ) then
    zeta  = 0.0
    ecz   = 0.0
    ilin  = 0
    iplas = 1

    c Set material data for this element integration point
    c for material type “PLASTIC_WB” (see "mater3.h”); all
    c other data remain unchanged from the original input.

    mattyp = 3

    do 10 i = 1, 3
      ee_ss_3(1,i) = e1_s1(1,i)
      ee_ss_3(2,i) = e1_s1(2,i)
    10 continue

    call MATSET ( mattyp,1,1 )

    write (not,100) iunit, ielt, eltip, kfab,
    &    mattyp, ee_ss_3(1,3)

  endif
endif

100 format ( ' USRFAB: iunit, ielt, eltip, kfab, mattyp, e36=',
    &    5i8, f14.8 )

Note that this coding specifies new properties for 8 elements in unit 7 of the user’s model—
calling MATSET because material properties for those 8 elements are changed.
USRFPF Failure Model

Subroutine USRFPF allows the STAGS user to implement his or her own first–ply failure model or to modify one of the existing failure models (which are implemented in the current version of STAGS in FPFi.F subroutines). The existing material degradation models (which are implemented in STAGS in DGRADi.F routines) can be utilized with the user’s USRFPF subroutine if they fit the user’s model. It will be necessary for the user to write his or her own USRDGD subroutine to implement a new material degradation model if the existing degradation models do not fit the new model.

Subroutine USRFPF is called [by STAGS’ Generic Constitutive Processor (GCP) facility] for each element that is fabricated with an orthotropic elastic brittle GCP material for which IFAIL = 99 on the l-10a (ORT_EL_BR_MATERIAL) record that defines that material.

As a template for adventuresome STAGS users, a failure model using the maximum strain criteria has been implemented in the USRFPF subroutine that is distributed with the program (and which is listed below for the reader’s convenience). The calling arguments to USRFPF are documented in this listing and will not be further described here.

**USRFPF Template**

c=deck    usrfpf

c=purpose Evaluate User-Written Failure Criterion

c=author  Norm Knight/Veridian

c=version January 2000

#include "keydefs.h"

* **************************
* * This routine is a template to guide the user in implementing a *
* * failure model in STAGS using the ORT_EL_BR_MATERIAL GCP material *
* * model -- where the failure selection flag IFAIL is set to 99 *
* **************************

#if __usage_
* CALLING SEQUENCE:
* call USRFPF ( iCMtyp, mpd, strain, stress, flag )
* INPUT ARGUMENT TYPE DESCRIPTION
* =============== ==== ===========
* iCMtyp         [I] Kinematic option for material processing
* mpd(*)        [R] Material property data
* strain(*)     [R] Mechanical strain vector
* stress(*)     [R] Stress vector
* OUTPUT ARGUMENTS
* ===============
* flag(*)        [I] Failure flags
*     ------------------------------------------------------------------

#include "con5.h"
#include "cs4xxx.h"
#include "pain3.h"
#include "stndcm.h"

*     I N T E R N A L  D E C L A R A T I O N S
*     ------------------------------------------------------------------
real      eps1C
real      eps1T
real      eps2C
real      eps2T
real      eps6
real      eps6F
real      index(6)
character mode*28
logical   NTITLE
integer   numcol

*     L O G I C
*     ------------------------------------------------------------------
*     Perform Failure Analysis using the Maximum Strain Criteria
if ( iCMtyp.eq.1 ) then
  1D CONTINUUM (ROD OR BAR ELEMENT)
  ================================
  numcol = 1
elseif ( iCMtyp.eq.2 ) then
  C0/C1 BEAM
  ===========
  numcol = 3
elseif ( iCMtyp.eq.3 ) then
  C0 SHELL
  =========
  numcol = 5
elseif ( iCMtyp.eq.4 ) then
  C1 SHELL (PLANE STRESS)
numcol = 3

Read strength data for ply

eps1C = mpd(19)
eps1T = mpd(20)
eps2C = mpd(21)
eps2T = mpd(22)
eps6F = mpd(23)

if ( NTITLE('X_Usrfpf') ) then
  write (not,910)  idelt, eltip, layer, layip, (flag(i),i=1,3)
  write (not,920)  strain(1), strain(2), strain(3)
endif

Compute failure indices:

if ( flag(1).eq.0 ) then
  Fiber tension failure:

  if ( NTITLE('X_Usrfpf') ) then
    write (not,930)  strain(1), flag(1), index(1), eps1T
  endif
  if (strain(1) .ge. 0.) then
    index(1) = strain(1)/eps1T
    if (index(1).ge.+1.0) then
      flag(1) = + istep
      mode = 'FIBER TENSILE FAILURE:  \\
      else
        flag(1) = 0
  endif
endif

  Fiber compression failure

  if (strain(1) .lt. 0.) then
    index(1) = strain(1)/eps1C
    if (index(1).le.-1.0) then
      flag(1) = - istep
      mode = 'FIBER COMPRESSIVE FAILURE:  \\
      else
        flag(1) = 0
  endif
endif
endif

if (flag(2).eq.0) then
  if ( NTITLE('X_Usrfpf') ) then
    write (not,940)  strain(2), flag(2), index(2), eps2T
  endif
  if (strain(2) .ge. 0.) then
    Matrix tension failure:
index(2) = strain(2)/eps2T
if (index(2).ge.+1.0) then
  flag(2) = + istep
  mode = 'MATRIX TENSILE FAILURE:     '
  if (nsp3.gt.0) write(not,900)
else
  flag(2) = 0
endif
endif
if (strain(2) .lt. 0.) then
  Matrix compression failure:
  index(2) = strain(2)/eps2C
  if (index(2).le.-1.0) then
    flag(2) = - istep
    mode = 'MATRIX COMPRESSIVE FAILURE:  '
    if (nsp3.gt.0) write(not,900)
  else
    flag(2) = 0
  endif
endif
if (flag(3).eq.0) then
  Fiber/matrix shear failure:
  if ( NTITLE('X_Usurf') )  then
    write (not,950)  strain(3), flag(3), index(3), eps6F
  endif
  eps6   = ABS(strain(3))
  index(3) = eps6/eps6F
  if (index(3).ge.+1.0) then
    flag(3) = + istep
    mode = 'INPLANE SHEAR FAILURE:      '
    if (nsp3.gt.0) write(not,900)
  else
    flag(3) = 0
  endif
  if ( NTITLE('X_Usurf') )  then
    write (not,960)  idelt, eltip, layer, layip, (flag(i),i=1,3)
  endif
elseif ( iCMtyp.eq.5 ) then
  2D PLANE STRAIN CONTINUUM
  numcol = 3
elseif ( iCMtyp.eq.6 ) then
  2D AXISYMMETRIC CONTINUUM
  numcol = 4
elseif ( iCMtyp.eq.7 ) then
* 3D CONTINUUM
* ============
numcol = 6
endif

900 format( a28,
  & ' ELDid=',i5,
  & '; ELTip=',i1,
  & '; Layer=',i3,
  & '; LayIP=',i1 )
910 format (' ELDid=',i5,
  & '; ELTip=',i1,
  & '; Layer=',i3,
  & '; LayIP=',i1,
  & '; BEGIN FAIL FLAGS=',3i2 )
920 format (' Input strain vector:',2x,e13.6,3x,e13.6,3x,e13.6 )
930 format (' DEBUG1 ***',5x,e13.6,i2,3x,e13.6,3x,e13.6 )
940 format (' DEBUG2 ***',5x,e13.6,i2,3x,e13.6,3x,e13.6 )
950 format (' DEBUG3 ***',5x,e13.6,i2,3x,e13.6,3x,e13.6 )
960 format (' ELDid=',i5,
  & '; ELTip=',i1,
  & '; Layer=',i3,
  & '; LayIP=',i1,
  & '; FINAL FAIL FLAGS=',3i4 )
end

The user may include data in addition to that which is transmitted to and from subroutine USRFPP through its calling sequence via STAGS’ user parameter input facility—using L-2a and/or L-2b records as required (for integers and real variables, respectively). This may be accomplished by including the upfi.h header file or (as is done here) by including the following explicit declarations and common statements:

integer userint
common / upi / userint(200)

and

real userflo
common / upf / userflo(200)

The cs4xxx.h header file that is included in this version of USRFPP contains four integer variables that are used to identify which material point is being examined:

idelt Element number within a unit
eltip Element surface integration point number
layer Layer number within the laminate
layip Through-the-thickness integration point within a layer

On entry into the USRFPP routine, a check is made to identify the kinematics option for the element by testing the value of the parameter iCMtyp. For C¹ shell elements, iCMtyp equals 3, and there are three stress values (σxx, σyy, τxy). Similarly there are three strain values. Next the strain
allowable values are loaded into local variables from the \texttt{mpd(*)} array (these are always the original input values).

The next step is to test to determine if failure in the fiber (or 1 direction) has occurred previously (if \texttt{flag(1)} is not zero) or that no failure has yet occurred (\texttt{flag(1)} equals zero). If \texttt{flag(1)} is not zero, then no further check is made for that mode and the process moves to \texttt{flag(2)} and \texttt{flag(3)} in a similar manner. If \texttt{flag(1)} equals zero, then the strain state is examined for failure (using the maximum strain criterion in this template). Tests are made for tension and compression independently. If failure is detected, then the alphanumeric variable \texttt{mode} is set to a 28-character string (users may change this value) that describes the failure mode for output purposes. If failure is detected, then \texttt{flag(1)} is set to a positive value for tensile failure or to a negative value for compressive failure. Shear failures are always set equal to a positive value. Typically, the magnitude of this value is the solution step number \texttt{istep} at which the failure was first detected.

Subroutine \texttt{USRFPF} is called for each layer integration point through the thickness of each layer in the laminate for each surface integration point of the element and for each element. It is a very low-level routine that is called many, many times; and if the user should choose to print output within this routine, he or she should expect a lot! Limited output control is provided through the integer parameter \texttt{nsp3} (which is the same as the \texttt{ILIST} parameter on the B-1 record of the \texttt{s2} input data. Setting that variable to a positive nonzero number will cause \texttt{USRFPF} (as well as other FPF* models) to print the failure mode and location.

Because \texttt{USRFPF} is a low-level routine, it is called during the evaluation of both the first and second variations during a nonlinear iteration. Hence multiple passes through this routine for each iteration may occur as a consistent state for the first and second variation is always sought.
USRLD General Loading

Subroutine USRLD is called once for each unit where IFLG = 1 on the associated Q-2 (shell unit) or U-2 (element unit) record. USRLD may define all of the loads in a given unit, or it may be used to complement loads defined on Q-3 (shell unit) or U-3 (element unit) records.

USRLD Structure

```
subroutine USRLD (iunit, X, Y, nrows, ncols, isys)

INTEGER
  :  iunit,     nrows,     ncols,     isys,
  :       lt,     ld,     li,     lj,
  :    lax
REAL
  : X(nrows),     Y(ncols),     P

***** Repeat for each load.       *****

  call FORCE ( P, lt, ld, li, lj, lax )

  return

end
```
## USRLD Data

### Input

<table>
<thead>
<tr>
<th>A</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>iunit</td>
<td>unit number</td>
<td>Unit number</td>
</tr>
<tr>
<td>isys</td>
<td></td>
<td>ISYS, Q-2 (p. 6-70)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ISYS, U-2 (p. 10-4)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>nrows</td>
<td></td>
<td>NROWS(i), i=iunit F-1</td>
</tr>
<tr>
<td>ncols</td>
<td></td>
<td>NCOLS(i), i=iunit F-1</td>
</tr>
<tr>
<td>X(nrows)</td>
<td></td>
<td>X  surface coordinate along each row</td>
</tr>
<tr>
<td>Y(ncols)</td>
<td></td>
<td>Y  surface coordinate along each column</td>
</tr>
</tbody>
</table>

✔ A data are always relevant.
✔ B data are relevant for shell units only.

### Output

<table>
<thead>
<tr>
<th>P</th>
<th>1t</th>
<th>1d</th>
<th>Shell unit: Q-3 (p. 6-70)</th>
</tr>
</thead>
<tbody>
<tr>
<td>li</td>
<td>lj</td>
<td>lax</td>
<td>Element unit: U-3 (p. 10-5)</td>
</tr>
</tbody>
</table>

$STAGSHOME$ /examples/usersub/usrld.F contains the code shown in “USRLD Structure”. See Section 12.3 “Example Problem” on page 12-66 for a practical example of USRLD.
**USRPT Nodes**

Subroutine USRPT is called once for each element unit where \( IUWP = 1 \) on the associated H-1 record. USRPT may define all of the nodes in a given element unit, or it may be used to complement nodes defined on S-1 records. Whether nodes are defined in USRPT or on S-1 records or in combination, they may be defined in any order, but when completely defined must be numbered continuously starting with 1. Thus, node numbering within each element unit is from 1 to \( n_S + n_U \), where \( n_S \) is the number of nodes defined on corresponding S-1 records, and \( n_U \) is the number of nodes defined in USRPT for the corresponding element unit.

**USRPT Structure**

```fortran
subroutine USRPT (iunit)

COMMON
: / NEWSY / xax, xay, xaz,
: yax, yay, yaz,
: zax, zay, zaz
INTEGER
: iunit, iupt,
: ius, irs, ics,
: iuvw, iruvw, iaux
REAL
: xax, xay, xaz,
: yax, yay, yaz,
: zax, zay, zaz,
: xg, yg, zg,
: gm

***** Repeat for each node. *****

call NODE (iupt, ius, irs, ics,
: xg, yg, zg,
: iuvw, iruvw, iaux, gm )

return
end
```
### USRPT Data

#### Input

<table>
<thead>
<tr>
<th>iunit</th>
<th>unit number</th>
</tr>
</thead>
</table>

#### Output

<table>
<thead>
<tr>
<th>iupt</th>
<th>ius</th>
<th>irs</th>
<th>ics</th>
<th>S-1 (p. 7-3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>xg</td>
<td>yg</td>
<td>zg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iuvw</td>
<td>iruvw</td>
<td>iaux</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>gm</th>
<th></th>
<th>U-4 (p. 10-7)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>COMMON/NEWSY/</th>
<th>xax</th>
<th>xay</th>
<th>xaz</th>
<th>S-2 (p. 7-8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>yax</td>
<td>yay</td>
<td>yaz</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

✔ **A, B** data are always defined.

✔ **C** data are relevant only when $iaux = 1$.

Note that $zax, zay, zaz$ are not set in USRPT; they are computed by **STAGS** (see S-2).

$STAGSHOME/examples/usersub/usrpt.F$ contains the code shown in “USRPT Structure”. See Section 12.3 “Example Problem” on page 12-66 for a practical example of USRPT.
**WALL** Shell Wall Fabrication Properties

User-written subroutine WALL is called at each integration point on the reference surface of each shell element where \( \text{IWALL} = 0 \) (M-5, T-3a, T-4a, USRELT). (For type E480 elements, subroutine WALL is called only once, at the element centroid.)

Subroutine WALL enables the user to define both geometric and material cross section properties as a function of position on the shell reference surface. This is effected by reference to the Wall Fabrication Table and/or by definition of data directly. As an example, the basic wall fabrication properties can be defined with a Wall Fabrication Table ID, and eccentricity can then be defined directly as a function of the position of each integration point on the shell reference surface.

**Shell Units**

Shell elements are numbered *locally* within each shell unit, from 1 to \( \text{NROWS} \times \text{NCOLS} \) (F-1), in row-major order.

Subroutine WALL is called for each shell element in each shell unit having \( \text{IWALL} = 0 \) (M-5).

**Element Units**

Elements are numbered sequentially as they are defined in each element unit, from 1 to \( n_T + n_U \), where \( n_T \) is the number of elements defined on T records and \( n_U \) is the number of elements defined in USRELT. This establishes *local* element numbering within each element unit. Subroutine WALL is called for each element-unit shell element for which \( \text{IWALL} = 0 \) (defined on a T-3, T-320, T-330, T-4, T-410, T-411 or T-480 record, or in USRELT).

The FORTRAN code shown in the following “WALL Template” listing is available in the $STAGSHOME/examples/usersub directory. The user must replace the lines that are printed in boldface type, on page 12-62, with coding that is appropriate for his or her problem.
WALL Template

c=deck wall
c=purpose User-written WALL subroutine
c/version April 2002

#include "keydefs.h"

#if _usage_
*     calling sequence: call WALL ( iunit, ielt, kelt, XYZg, XYs,
*                                   zeta, ecz, ilin, iplas )
*     Input Arguments:
*     ================
*     iunit = unit number
*     ielt  = local element number (in unit iunit)
*     kelt  = element type code
*     XYZg  = \{x,y,z\} global coordinates
*     XYs   = \{s,t\} surface coordinates (shell unit, only)
*     Output Arguments:
*     =================
*     zeta  = zeta (see M-5 or T-3 for details)
*     ecz   = eccentricity (see M-5 or T-3 for details)
*     ilin  = nonlinearity flag
*     iplas = plasticity flag
*     
#endif

******************************************************************************
subroutine WALL ( iunit, ielt, kelt, XYZg, XYs,
&                        zeta, ecz, ilin, iplas )
******************************************************************************

_implicit_none_

Integer    iunit
Integer    ielt
Integer    kelt
Integer    ilin
Integer    iplas
Real       XYZg(3)
Real       XYs(2)
Real       zeta
Real       ecz

Integer         maxLAY
PARAMETER     ( maxLAY = 100 )
WALL Template (continued)

Integer         maxSM
PARAMETER     ( maxSM  = 3 )

Integer         nit,  not
common /nitnot/ nit,  not

Integer         itaw, kwall, nlay, nlip, nsmrs
common /WALLX / itaw, kwall, nlay, nlip, nsmrs

Integer         matL (maxLAY)
Real            tL   (maxLAY)
Real            zetL (maxLAY)
Integer         lsoL (maxLAY)
Real            e1L  (maxLAY)
Real            u12L (maxLAY)
Real            gL   (maxLAY)
Real            rhoL (maxLAY)
Real            e2L  (maxLAY)
Real            a1L  (maxLAY)
Real            e2L  (maxLAY)
common /WALL1 / matL, tL,   zetL, lsoL, e1L, u12L,
 &                gL,   rhoL, a1L,  e2L,  a2L

Integer         matF, matM
Real            ttL   (maxLAY)
Real            xxL  (maxLAY)
Real            zetwL (maxLAY)
Real            oL   (maxLAY)
Real            eF,   uF,   rhoF,  alF
Real            eM,   uM,   rhoM,  alM
common /WALL2 / matF, matM,
 &                ttL,  xxL,  zetwL, oL,
 &                eF,   uF,   rhoF,  alF,
 &                eM,   uM,   rhoM,  alM

Integer         matC, matS
Real            ct,   cc,   ch,   cd,   cb
Real            ts,   phi,  anc
Real            eC,   uC,   rhoC,  alC
Real            eS,   uS,   rhoS,  alS
common /WALL3 / matC, matS,
 &                ct,   cc,   ch,   cd,   cb,
 &                ts,   phi,  anc,
 &                eC,   uC,   rhoC,  alC,
 &                eS,   uS,   rhoS,  alS
WALL Template (continued)

Integer         ta,   mat,  itvs, idumt
Real            ccc,        cts
common /WALL4 / ta,   mat,  itvs, idumt,
&                ccc(6,6),   cts(2,2)

Integer         icroSM (maxSM)
Real            spaSM  (maxSM)
Real            zetSM  (maxSM)
Real            xsiSM  (maxSM)
Real            eczSM  (maxSM)
common /SMEAR / icroSM, spaSM, zetSM, xsiSM, eczSM

write (not,900)
900 format ('' SUBROUTINE WALL HAS NOT BEEN PROVIDED.'')

STOP
end
WALL Data

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iunit</td>
<td>unit number</td>
</tr>
<tr>
<td>ielt</td>
<td>unit local element number</td>
</tr>
<tr>
<td>kelt</td>
<td>shell element code; e.g., 410</td>
</tr>
<tr>
<td>XYZg(3)</td>
<td>(x, y, z) global coordinates</td>
</tr>
<tr>
<td>XYs(2)</td>
<td>(X, Y) surface coordinates; relevant for shell units only.</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>zeta</td>
<td></td>
</tr>
<tr>
<td>ecz</td>
<td></td>
</tr>
<tr>
<td>ilin</td>
<td></td>
</tr>
<tr>
<td>iplas</td>
<td></td>
</tr>
</tbody>
</table>

Shell units:
- M-5 (p. 6-24)
- T-3 (p. 8-14)
- T-4 (p. 8-17)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMON/WALLX/itaw</td>
<td></td>
</tr>
<tr>
<td>kwall</td>
<td></td>
</tr>
<tr>
<td>nlay</td>
<td></td>
</tr>
<tr>
<td>nlip</td>
<td></td>
</tr>
<tr>
<td>nsmrs</td>
<td></td>
</tr>
</tbody>
</table>

K-1 (p. 5-129)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMON/WALL1/matL</td>
<td></td>
</tr>
<tr>
<td>tL</td>
<td></td>
</tr>
<tr>
<td>zetL</td>
<td></td>
</tr>
<tr>
<td>lsoL</td>
<td></td>
</tr>
<tr>
<td>e1L</td>
<td></td>
</tr>
<tr>
<td>u12L</td>
<td></td>
</tr>
<tr>
<td>gL</td>
<td></td>
</tr>
<tr>
<td>rhoL</td>
<td></td>
</tr>
<tr>
<td>a1L</td>
<td></td>
</tr>
<tr>
<td>e2L</td>
<td></td>
</tr>
<tr>
<td>a2L</td>
<td></td>
</tr>
</tbody>
</table>

K-2 (p. 5-133)

Elastic material properties:

For each layer i:
- Define matL(i)
- If and only if matL(i) = 0
  - Define E1, ν12, G, ρ, α1, E2, α2
  - For layer i
### Restrictions on variation of wall properties

\( \text{ilin, iplas (A data) and } \text{kwall, nlip (B data)} \) must be constant within each element.

<table>
<thead>
<tr>
<th>COMMON/WALL2/</th>
<th>COMMON/WALL3/</th>
<th>COMMON/WALL4/</th>
<th>COMMON/SMEAR/</th>
</tr>
</thead>
<tbody>
<tr>
<td>matF matM</td>
<td>matC matS</td>
<td>ta mat itvs</td>
<td>microSM spaSM</td>
</tr>
<tr>
<td>ttL xxL</td>
<td>ct cc ch</td>
<td>ccc(6,6)</td>
<td>zetSM</td>
</tr>
<tr>
<td>zetwL</td>
<td>cd cb</td>
<td>cts(2,2)</td>
<td>xsiSM eczSM</td>
</tr>
<tr>
<td>oL eL eF uF</td>
<td>ts phi anc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>alF eM uM</td>
<td>eC uC rhoC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rhoF alM</td>
<td>alC eS uS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>rhoS alS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Fiber elastic material properties:**
  - when \( \text{matF} = 0 \), define \( E, v, \rho, \alpha \)
- **Matrix elastic material properties:**
  - when \( \text{matM} = 0 \), define \( E, v, \rho, \alpha \)

- **Corrugation elastic material properties:**
  - when \( \text{matC} = 0 \), define \( E, v, \rho, \alpha \)
- **Skin elastic material properties:**
  - when \( \text{matS} = 0 \), define \( E, v, \rho, \alpha \)

✔ All data are always defined.
When plasticity is included (\(i\text{plas} \geq 1\)), all data must be constant within each element.

If either of the two above restrictions is violated by \textsc{wall} (called at each integration point), \textsc{stags} will resolve the discrepancies by resetting conflicting data to the values given for integration point 1.

**Material properties**

Linear elastic material properties may be defined in C, D, and E data. However nonlinear material properties may be defined indirectly only, by reference to a Wall Fabrication Table entry (see “Wall fabrication table selection” on page 12-65).

**Wall fabrication table selection**

- Set \(i\text{taw} > 0\). The remaining B data are irrelevant.

- **Layered wall**
  
  Set \(i\text{taw} = 0\) and \(k\text{wall} = 1\). Define B, C data.

- **Fiber reinforced wall**
  
  Set \(i\text{taw} = 0\) and \(k\text{wall} = 2\). Define B, D data.

- **Corrugation stiffened wall**
  
  Set \(i\text{taw} = 0\) and \(k\text{wall} = 3\). Define B, E data.

- **General wall**
  
  Set \(i\text{taw} = 0\) and \(k\text{wall} = 4\). Define B, F data.

- **Smeared stiffeners**
  
  When \(i\text{taw} = 0\) only: set \(n\text{smrs} > 0\) and define G data.

  Material behavior is restricted to linear elasticity in walls containing smeared stiffeners.
12.3 Example Problem

Figure 12.1 contains a sketch of the example problem used to illustrate the user-written subroutine concept. The structure is a quarter cylinder, bounded circumferentially by $\theta = 0^\circ$ and $\theta = 90^\circ$, and axially by $x_g = 0$ and $x_g = L$. The reference surface is defined by the expressions $y_g = R\sin\theta$, $z_g = R\cos\theta$. The transformation between $(x_g, y_g, z_g)$ global coordinates and $(x_a, y_a, z_a)$ nodal-auxiliary coordinates is indicated. Boundary conditions and loads are defined in the nodal-auxiliary coordinate system. The nodal load $P$ is shown acting in the positive $u$ direction, and the normal pressure $q$ is shown acting in the positive $w$ direction. Negative values for $P$, $q$ will result in loading acting in the negative $u$, $w$ directions, respectively. Note that on the $x_g = 0$ boundary, in addition to the symmetry boundary condition, the constraint $u = c$, where $c$ is a constant, is prescribed. The effect of requiring uniform axial displacement along the $x_g = 0$ boundary is that the discrete nodal load $P$ acts like a line load $p = P/(\pi R/2)$ on that edge.

Just as the loading is parameterized by $P$ and $q$, the geometry is parameterized by the radius, $R$, and the length, $L$. Two additional parameters, $nrows$ and $ncols$, the number of rows and columns, control discretization. A $9 \times 9$ mesh is shown in Figure 12.1.

The four user-written subroutines employed to model this quarter cylinder are:

- **USRPT**: defines nodal locations, boundary conditions, and nodal-auxiliary coordinate systems
- **USRELT**: defines quadrilateral shell elements
- **USRLD**: defines the loading, $P$ and $q$
- **UCONST**: defines the $u = c$ constraint on the $x_g = 0$ boundary

This model could be further parameterized by replacing $L$ with $x_g^1$ and $x_g^2$, thereby permitting a cylinder of axial length $x_g^1 \leq x_g \leq x_g^2$. Adding two additional parameters, $\theta^1$ and $\theta^2$, would allow a cylinder of arclength $\theta^1 \leq \theta \leq \theta^2$. Minor modifications to USRPT are all that would be necessary. Similarly, the element type could be added as a parameter, requiring only minor modifications to USRELT. The wall fabrication could be parameterized in a user-written subroutine WALL, and so forth.

The **INP** and **BIN** input files are shown next, followed by listings of the FORTRAN code for the user-written subroutines.

The input files, along with files containing source code for the user-written subroutines, are found in $\$STAGSHOME/examples/cylinder$. 
User-written subroutines—cylindrical shell example problem.

\[ 0 \leq x_g \leq L \quad 0 \leq \theta \leq 90^\circ \]
\[ y_g = R \sin \theta \quad z_g = R \cos \theta \]

boundary conditions:  
\[ x_g = 0: \text{ simple support} \quad \& \quad u = \text{constant} \]
\[ x_g = L: \text{ clamped} \]
\[ \theta = 0^\circ, 90^\circ: \text{ symmetry} \]

\[
\begin{bmatrix}
  x_a \\
  y_a \\
  z_a
\end{bmatrix} =
\begin{bmatrix}
  1 & 0 & 0 \\
  0 & \cos \theta & -\sin \theta \\
  0 & \sin \theta & \cos \theta
\end{bmatrix}
\begin{bmatrix}
  x_g \\
  y_g \\
  z_g
\end{bmatrix}
\]

Figure 12.1  User-written subroutines—cylindrical shell example problem.
cylinder.inp

Quarter Cylinder, User-Written Subroutines $ A-1
0 $ B-1
0 1 0 0 0 8 $ B-2 UCONST
1 0 1 1 $ B-3
0 0 0 0 0 1 1 $ H-1 USRPT USRELT
1 $ I-1
10.0E6 0.3 $ I-2
1 1 1 $ K-1
1 0.1 0.0 1 $ K-2
2 4 $ L-1

$ ------------------------------------------ $
$      rows      cols                        $
$      ----      ----                        $
9         9                         $ L-2A discretization
$ ------------------------------------------ $
$ R         L         P         q     $
$ ---       ---       ---       ---    $
6.4      10.0      1.5E5    -100.0   $ L-2B geometry, loads
$ ------------------------------------------ $
1 $ U-1
1 0 1 $ U-2 USRLD
1 $ V-1

cylinder.bin

Quarter Cylinder, User-Written Subroutines $ A-1
0 1 $ B-1
1.0 $ C-1
SUBRPT

subroutine USRPT (iunit)

COMMON
:: / UPI   / nrows, ncols, UserInt(198)
:: / UFP   / R, L, P, q, UserFlo(196)
:: / NEWSY / xax, xay, xaz,
:: yax, yay, yaz
INTEGER
:: iunit, nrows, ncols, UserInt,
:: iupt, ius, irs, ics,
:: iuvw, iruvw, iaux,
:: i, j
REAL
:: R, L, P, q,
:: xax, xay, xaz, UserFlo,
:: yax, yay, yaz,
:: xg, yg, zg, gm,
:: delX, delY, pi,
:: theta, Ct, St

iupt = 0
ius  = 0
irs  = 0
ics  = 0
iaux = 1
gm   = 0

xax = 1
xay = 0
xaz = 0
yax = 0

pi  = 4 * ATAN(1.)
delX = L/(nrows-1)
delY = (pi/2)/(ncols-1)

DO 20 j = 1,nrows
   xg = (j-1)*delX
   DO 10 i = 1,ncols
      theta = (i-1)*delY
      Ct    = COS(theta)
      St    = SIN(theta)
      yg    = R*St
      zg    = R*Ct
      yay   = Ct
      yaz   = -St
   10      CONTINUE
   if (j .eq. 1) then
      if ((i .eq. 1) .or. (i .eq. ncols)) then
         iuvw = 100
         iruvw = 010
      else
         iuvw = 100
         iruvw = 011
      endif
   elseif (j .eq. nrows) then
      iuvw = 000
   endif

20 CONTINUE
iruvw = 000
elseif ((i .eq. 1) .or. (i .eq. ncols)) then
    iuvw  = 101
    iruvw = 010
else
    iuvw = 111
    iruvw = 111
endif
iupt = iupt + 1
    call NODE (iupt, ius, irs, ics,
    :           xg,   yg,   zg,
    :             iuvw, iruvw,iaux, gm )
10  continue
20  continue

    return
end
USRELT

subroutine USRELT (iunit)

COMMON
:/ UPI / nrows, ncols, UserInt(198)
INTEGER
:maxN
PARAMETER
:( maxN = 9 )
INTEGER
:iunit, nrows, ncols, UserInt,
: node(maxN), kelt, iwall,
: ilin, iplas, integ, ipenl,
: iang, n0, i, j
REAL
: zeta, ecz,
: rx, ry, rz

kelt = 410
iwall = 1
zeta = 0
ecz = 0
ilin = 0
iplas = 0
integ = 0
ipenl = 0
iang = 0

do 20 j = 1,nrows-1
    n0 = (j-1)*ncols
    do 10 i = 1,ncols-1
        node(1) = n0 + i
        node(2) = node(1) + ncols
        node(3) = node(2) + 1
        node(4) = node(1) + 1
        call SHELL (node, kelt, iwall, zeta, ecz,
        : ilin, iplas, integ, ipenl,
        : iang, rx, ry, rz)
   10    continue
20  continue

return
end
USRLD

subroutine USRLD (iunit, X, Y, nrows, ncols, isys)

COMMON
: / UPF / R, L, P, q, UserFlo(196)
: : iunit, nrows, ncols, isys,
: : lt, ld, li, lj,
INTEGER
: : lax
REAL
: : X(nrows), Y(ncols), UserFlo,
: : R, L, P, q

lt = 1
ld = 1
li = 1
lj = 0
lax = 0

call FORCE (P, lt, ld, li, lj, lax)

lt = 4
ld = 3
li = 0

call FORCE (q, lt, ld, li, lj, lax)

return
end
**UCONST**

subroutine UCONST

INTEGER      maxN
PARAMETER ( maxN = 2 )

INTEGER iu(maxN)
INTEGER ix(maxN), iy(maxN), iz(maxN)
INTEGER id(maxN)
REAL    cc(maxN)

INTEGER      nrows, ncols, UserInt
COMMON / UPI / nrows, ncols, UserInt(198)

INTEGER i
INTEGER nterms

nterms = 2

iu(1) = 1
iu(2) = 1

ix(1) = 1

iy(1) = 0
iy(2) = 0

iz(1) = 1
iz(2) = 1

id(1) = 1
id(2) = 1

cc(1) =  1.0E7
cc(2) = -1.0E7

do 10 i = 2, ncols
   ix(2) = i
   call CONSTR ( nterms, iu, ix, iy, iz, id, cc )
10 continue

return
end
13
User–Defined Elements

13.1 Introduction

The User element (UEL) feature in STAGS 5.0 gives advanced analysts and developers powerful new tools for solving problems that lie beyond the scope of those that STAGS has traditionally treated. The process by which User elements are defined in STAGS is described concisely here and fully in Chapter 9 of the STAGS Elements Manual. The input records with which User elements may be employed in constructing a STAGS model are described in Section 9.8 on page 9-163.

As noted in Section 9.8, there are two distinct (but strongly coupled) operations that must be performed to define a User element for use in the construction of STAGS models and for use in conducting analyses with those models.

The first operation (which must be performed in generating the INP file that STAGS’ s1 processor reads and processes to construct the STAGS model for a given problem) is the analyst’s definition of top-level information about the User element—including (but not limited to) the element-type identifier by which it is to be referenced, the number of node points required to specify any given element of this type, the names and characteristics of data associated with all elements of this type, etc. This is done by including a set of user-element-definition directives (see Sections 13.2 and 13.3) at any appropriate point in the analyst’s INP file prior to any input records that command the inclusion of one or more User elements of that type in the model being constructed.

The second operation (which must be performed prior to executing any STAGS processor) is the analyst’s (developer’s) generation of all user-element-specific FORTRAN- and C-language routines as described in Section 13.4. These routines may contain explicit definitions of the element features (internal force vector, stiffness matrix, stress and strain recovery, etc.) or they may contain CALL statements to other subroutines or functions that compute the various features.
User-Defined Elements

STAGS processors need these routines to construct a STAGS model in which these user-defined elements are employed and to perform analysis and postprocessing operations with that model—and the successful linking of each of these user-element-enhanced processors.

Before getting into either of these operations, it is appropriate to pause briefly and note that the principal goal of the STAGS User element framework is to provide a set of features that enable element researchers and advanced or adventurous analysts to add new element types to STAGS without having to be overly concerned with the gory and sometimes excruciating details of the STAGS element integration process—focusing only on the development of their new elements.

To provide an almost transparent element integration environment, the STAGS User element framework provides the following features:

- Arbitrary User element types are represented in the standard STAGS element description scheme as element types $900 \leq \text{type} \leq 999$.
- Any number (less than or equal to one hundred) of User element types and their associated data descriptions may be specified.
- Any number of User elements of any UEL type may be defined in STAGS models.
- Each User element type definition specifies a set of named data that are available for every element of that type.
- User element program code can retrieve and store the values of individual User element data items by their names, or can access an element's integer and floating-point data blocks in the ABAQUS style.
- An arbitrary number of user-described property sets can be defined that specify a collection of named data that may be arbitrarily associated with one or more elements of any user-defined type.
- User element wrapper routines are invoked automatically by STAGS as required by a solution process. Users implementing new elements need not be concerned with the details of the STAGS solution process.

An arbitrary number of user-defined element types can be specified using a free field description of the nodal and data specifications for each element. These descriptions are entered directly into the STAGS model input file and provide the user the ability to describe the element type and its data requirements in any manner that facilitates the most natural ordering. Comments and flexible line spacing are allowed for enhanced readability.

Each User element type description may specify a collection of named integer and/or floating-point variables associated with each User element defined in a STAGS model. Naming User element variables greatly increases a user's understanding of both simple and complex datasets and reduces the likelihood of confusing data values required by different element types. This is
a general capability; the developer or researcher is not restricted by the kind or amount of data that can be associated with each User element type.

The User element framework also introduces a new STAGS feature called *User property sets*. A developer or researcher may define any number of property sets, each of which is identified by a meaningful name and a unique numeric identifier. The property items in a property set, entered by the user in free field format (much like User element type descriptions), collect integer and/or floating-point data that are best described when grouped together. This feature provides a general and flexible technique for describing data required by any new capabilities added by developers or researchers. The creation of new, user-defined element types is just one example.

### 13.2 User Element Definition Directives

Returning to the subject of *directives* for definition of User elements, we note that each UEL type is described in the *INP* file for a STAGS model by a set of four required directives and two optional directives (printed in square brackets) that are ordered as follows:

*userElement*... — initiate userElement specifications
*    *dofOrdering*... — specify number and types of DOF at each node
*    *nodeSequence*... — specify nodal sequence for transformations
[*floatVariables*...] — specify float-type variables (if any)
[*integerVariables*...] — specify integer-type variables (if any)
*end userElement* — terminate userElement specifications

Directive formatting is free-field. String values that contain one or more spaces must be enclosed in matching single or double quotes. Directive names and directive attribute names are case-insensitive. Empty (blank) lines and lines containing only $-initiated comments are ignored during parsing of directives.

The *floatVariables* data groups are optional. They may appear in any order and may be repeated as many times as required. All float variable items are collected sequentially as they are defined and are placed in a single logical group of data type float.

The *integerVariables* data groups are also optional. They may appear in any order and may be repeated as many times as required. All float variable items are collected sequentially as they are defined and are placed in a single logical group of data type float. All integer variable items are collected sequentially as they are defined and are placed in a single logical group of type integer.
**Example of User element definition**

Rather than going through a long-winded presentation of directive syntax and conventions, let us simplify this by looking at a straightforward example that shows how a user might define a User element—to be referenced as a type 901 element—for a simple three-noded beam. The end nodes of this beam are identified as node number 1 and node number 2, and the internal node is identified as node number 3. A fourth node is associated with the element in order to compute the element’s orientation matrix. Associated with this element type is a set of named floating-point variables and a set of named integer variables. These named data will be defined for each user-defined element created by STAGS and can be manipulated by developers and by the element researcher using utility routines that are described briefly below and in greater detail in Chapter 9 of the STAGS Elements Manual.

```plaintext
*userElement name = "Beam Element"  type = 901  nodes = 4

*dofOrdering
$  Node  DOF...
$  ----------------
  1    1 2 3 4 5 6
  2    1 2 3 4 5 6
  3    1 2 3 4 5 6
  4    0

*nodeSequence
$  Nodes...
$  --------
  1 2 4

*floatVariables
$  Name  Size
$  ----------------
  MaxStress      1
  MaxStressLoc   3
  SectionData    5
  ShearFactor    1
  OtherStuff    15

*integerVariables
$  Name  Size
$  ----------
  NIPS     1
  List     10
```

*end userElement
Notes about this User element definition example

The following points about this example are generally true for all User element type definitions:

- User element type names must have string values and are used for descriptive purposes only.
- User element type names are limited to 40 characters in length.
- User element type numbers must be in the range $900 \leq \text{type} \leq 999$, inclusive and must be unique (i.e., a UEL type must be defined only once).
- A User element type may have any number of nodes.
- The DOF ordering for each node must be specified in a single group.
- All nodes are expected to have zero (0), three (3) or six (6) degrees of freedom, with DOFs defined in agreement with STAGS’s DOF-ordering convention:

  1 : x–displacement
  2 : y–displacement
  3 : z–displacement
  4 : x–rotation
  5 : y–rotation
  6 : z–rotation

- Element reference nodes are specified as having zero degrees of freedom (as for node 4 in this example).
- Nodes used to define an element’s orientation matrix are identified by a single, required *nodeSequence directive. Either three (3) or four (4) nodes in the *dofOrdering group are referenced by index in this directive. If three nodes are referenced, the element’s x–axis is directed from the first to the second node. The element’s z–axis is directed as the vector cross product of the x–axis and the vector from the first to the third node. The element’s y–axis follows from the right-hand rule with respect to the x–axis and the z–axis. If four nodes are referenced, the element’s x–axis is directed from the first to the second node. The element’s z–axis is directed as the vector cross product of the vector from the first to the third node and the vector from the second to the fourth node. The element’s y–axis follows from the right-hand rule with respect to the x–axis and the z–axis.
- User element variable names are limited to 40 characters in length.
- Variable names containing one or more spaces must be enclosed in matching single or double quotes.
- Variable names are case-insensitive for comparison and must be unique within their User element type definition.
- A variable’s size (logical length) must be specified and must be an integer number greater than zero.
13.3 User Property Set Definition Directives

User property sets are described (during STAGS s1 processing) through a set of directives as shown in the following example. Directives defining a User property set are ordered as follows:

*userProperty... – initiate userProperty specifications
[*floatProps...] – define one or more float-type data parameters
[*integerProps...]– define one or more integer-type data parameters
*end userProperty – initiate userProperty specifications

As with the previously-described user-element–definition directives, empty (blank) lines and lines containing only $-initiated comments are ignored during parsing of directives; and directive names and directive attribute names are case-insensitive. Directive formatting is free-field. String values that contain one or more spaces must be enclosed in matching single or double quotes.

The *floatProps data group is optional; *floatProps data groups may appear in any order and may be repeated as many times as required. All float property items are collected sequentially as they are defined and are placed in a single logical group of data type float. The *integerProps data group is optional when the analyst/developer chooses not to use STAGS’ GCP processor to specify material properties and fabrications; it is mandatory when he/she uses the GCP. When needed, *integerProps data groups may appear in any order and may be repeated as many times as required. All integer property items are collected sequentially as they are defined and are placed in a single logical group of data type integer.

Example # 1 of User property set definition

*userProperty  name = "Beam Element"  id = 901
  *integerProps
    $  Required Standard Data
    $  ----------------------
    ActiveNodes    2
    SamplingCount  10
    StrainCount    6
    StressCount    6
  *end userProperty

*userProperty  name = "Aluminum 6061-T6"  id = 6061
  *floatProps
    E                        10.00e+6
    G                         3.75e+6
    MassDensity               2.54e-4
    PoissonRatio              0.3
    TensileUltimateStrength  38.00e+3
    TensileYieldStrength     35.00e+3
    CompresiveYieldStrength  35.00e+3
    ShearYieldStrength       20.00e+3
    ThermalExpansion       1.30e-5
  *end userProperty
This example builds on the previous example that showed how an element researcher might define a new element type for a three-noded beam with a single internal node. In addition to the element type’s set of named integer and floating-point element variables, there may be several User property sets defined to complete the specification of an element. These property sets contain collections of related property items and can be logically associated with any number of elements or referenced from any number of appropriate contexts. User property sets can be manipulated by developers and element researchers using utility routines that are described briefly below and more completely in Chapter 9 of the *STAGS Elements Manual*.

As noted above, the **integerProps** data group is mandatory when the analyst/developer employs STAGS’ GCP processor to specify material properties and fabrications. Under these circumstances, two integer GCP-interface parameters (Class and Kintype) must be specified in addition to the four other parameters that are typically required (ActiveNodes, SamplingCount, StrainCount and StressCount). In the current versions of STAGS, the Class and Kintype parameters may have the following values:

- **Class** specifies class type for the element(s) in question:
  - = 0 — not specified (do not use the GCP interface)
  - = 1 — beam (1-D) elements
  - = 2 — shell (2-D) elements
  - = 3 — solid (3-D) elements

- **Kintype** specifies the kinematic type for the element(s) in question:
  - = 0 — not specified (do not use the GCP interface)
  - = 1 — type C^0 (qC0) elements — isoparametric (Mindlin) shell elements with shear (and usually no natural drill freedom); examples include the MIN4 and E480 (ANS) quads; beams of this type have 8 resultant components; shells have 6
  - = 2 — Kirchoff elements with C^1 (qC1) compatibility; beams of this type have 4 resultant components; shells have 6
  - = 3 — 3-D (solid) elements (q3D), with 6 stresses
The following example shows a User property set that might be used to define some of the integer-type properties of a type-900 user element—a quadrilateral $C^0$ shell element, in this case—when the the GCP processor is used to specify its material properties and fabrications:

**Example # 2 of User property set definition**

```
*userProperty  name = "MIN4 Element"  id = 900
    *integerProps
        $  Required Standard Data
        $  ----------------------
        ActiveNodes     4
        SamplingCount   4
        StrainCount     8
        StressCount     8
        $  GCP-interface Data
        $  ------------------
        Class           2
        Kintype         1
    *end userProperty
```

The following points made about these two User property set definition examples are generally true for all such definitions:

- For each User element type definition there must be one property set with an identifier (id) that matches the identifier of the type definition. In addition to any other properties this property set may contain, the following integer property items must be specified: `ActiveNodes` (the number of active nodes for the element type), `SamplingCount` (the number of locations where stresses and strains will be evaluated), `StrainCount` (the number of strain components associated with the element type), `StressCount` (the number of stress components associated with the element type). Failure to provide these property items in a property set associated with an element type definition will lead to a run-time error.
- User property set names must have string values and must be unique across all User property sets.
- User property set names are limited to 40 characters in length.
- User property set identifiers (IDs) must have numeric values and must be unique across all User property sets.
- Property item names containing one or more spaces must be enclosed in matching single or double quotes.
- Property item names are limited to 40 characters in length.
- Property item names are case-insensitive for comparison and must be unique within their property set definition.
13.4 User Element Model–Definition Routines

When one or more types of User elements are to be employed in a STAGS model, it is sometimes necessary for the analyst (or developer) to generate a user-element-enhanced version of STAGS’ s1 (model–definition) processor. It is always necessary for him (or her) to generate User–element–enhanced versions of STAGS’ s2 (analysis) processor and of any of the STAGS-system post-analysis processor(s) that are to be used with that model. The broad outlines of how this may be done are described in the following paragraphs. The interested reader should consult Chapter 9 of the STAGS Elements Manual for more information about this.

Top–level model–definition routines

When STAGS’ s1 processor encounters a User element, s1 calls a number of generic User–element definition routines to perform operations that are required universally. s1 also calls upon three other user-element “dispatch” routines to perform additional, nonstandard model-definition operations for that User element—as and if they are required. The primary function of each of these dispatch routines is evident by its name: UelEltDef, UelLoadDef and UelPressDef. Each of these dispatch routines calls one or more next–level program- or developer-supplied routines that perform nonstandard definition operations that are appropriate for each type of User element. The “program-supplied” qualifier is appropriate when special operations are not required; the “developer-supplied” qualifier applies when special operations are required.

For reference purposes, the current default version of the UelEltDef dispatch routine is shown in the FORTRAN listing beginning on the next page. The current default versions of subroutines UelLoadDef and UelPressDef are listed in Chapter 9 of the STAGS Elements Manual. The names of the three user-element-type-specific routines that this version of UelEltDef calls—UelEltDef900, UelEltDef901 and UelEltDef902, for user-element types 900, 901 and 902, respectively—are completely arbitrary and may be changed to reflect the user’s particular choice (subject to the architectural constraint that each User element type must have a unique identifier number that lies in the range from 900 to 999, inclusive). A do-nothing “starter” template is provided for each of the three particular subroutines that this template version of UelEltDef calls. The default version of the UelEltDef900 routine that this version of UelEltDef calls is shown in the listing following that of UelEltDef. All three top-level s1-processor dispatch routines and their respective type-specific children are documented in Chapter 9 of the STAGS Elements Manual.
subroutine UelEltDef

1: c=deck UelEltDef
2: c=purpose Perform definition operations for a user-defined element
3: c=author -----------------------------------------------
5: c=author ------------------- Ph: (858)259-2773 or (858)481-9907
6: c=version January 2005
7:
8: #if _usage_
9: *
10: * call UelEltDef (idef, type, unit, uid,
11: * ifab, iang, ilin, iplas,
12: * nnodes, nodseq, triad, unodes,
13: * nodes, maxdof, nfvars, nivars,
14: * t0, xg, xe)
15: *
16: * Input Arguments
17: *
18: * idef = current User element definition index
19: * type = element type: >=900
20: * unit = current unit number
21: * uid = user specified element number
22: * ifab = fabrication code
23: * iang = material angle flag
24: * ilin = linearity flag
25: * iplas = plasticity flag
26: * nnodes = number of nodes
27: * nodseq = nodal sequence for element frame
28: * triad = material orientation matrix
29: * unodes = element nodes (user-definition)
30: * nodes = element nodes (STAGS-definition)
31: * maxdof = nodal maxdof vector
32: * nfvars = number of user float variables
33: * nivars = number of user integer variables
34: * t0 = element orientation matrix
35: * xg = global nodal coordinates
36: * xe = local nodal coordinates
37: *
38: #endif
39:
40: ************************************************************************
41:           Subroutine UelEltDef (idef, type, unit, uid,
42:      &                          ifab, iang, ilin, iplas,
43:      &                          nnodes, nodseq, triad, unodes,
44:      &                          nodes, maxdof, nfvars, nivars,
45:      &                          t0, xg, xe)
46: ************************************************************************
47:
48: #     include   "keydefs.h"
49:
50: _implicit_none_
subroutine UelEltDef (continued)

51:
52:  #     include   "stndcm.h"
53:
54:  Integer    idef
55:  Integer    type
56:  Integer    unit
57:  Integer    uid
58:  Integer    ifab
59:  Integer    iang
60:  Integer    ilin
61:  Integer    iplas
62:  Integer    nnodes
63:  Integer    nodseq(4)
64:  _float_    triad(3,3)
65:  Integer    unodes(nnodes)
66:  Integer    nodes(nnodes)
67:  Integer    maxdof(nnodes)
68:  Integer    nfvars
69:  Integer    nivars
70:  _float_    t0(3,3)
71:  Real       xg(3,nnodes)
72:  _float_    xe(3,nnodes)
73:
74:  Integer    a
75:  Integer    b
76:  Integer    m
77:  Character  msg*44
78:  Logical    NTITLE
79:
80:  if (NTITLE('X_EltDef')) then
81:     m = min(nnodes,20)
82:     print 100, idef,   type,   unit,   uid,
83:     &              ifab,   iang,   ilin,   iplas,
84:     &              nnodes, nodseq, nfvars, nivars,
85:     &              triad
86:     print 200, (unodes(a),a=1,m)
87:     print 300, (nodes (a),a=1,m)
88:     print 400, (maxdof(a),a=1,m)
89:     print 500, ((t0(a,b),a=1,3),b=1,3)
90:     print 600, ((xg(a,b),a=1,3),b=1,m)
91:     print 700, ((xe(a,b),a=1,3),b=1,m)
92:     endif
93:  *
94:  PERFORM DEFINITION OPERATIONS FOR A USER-DEFINED ELEMENT
95:  *
96:  =====================================================================
subroutine UelEltDef (continued)

100:       if (type.eq.900) then
101:
102:          call UelEltDef900 (idef, type, unit, uid,
103:           &                      ifab, iang, ilin, iplas,
104:           &                      nnodes, nodseq, triad, unodes,
105:           &                      nodes, maxdof, nfvars, nivars,
106:           &                      t0,     xg,     xe)
107:
108: elseif (type.eq.901) then
109:
110:          call UelEltDef901 (idef, type, unit, uid,
111:           &                      ifab, iang, ilin, iplas,
112:           &                      nnodes, nodseq, triad, unodes,
113:           &                      nodes, maxdof, nfvars, nivars,
114:           &                      t0,     xg,     xe)
115:
116: elseif (type.eq.902) then
117:
118:          call UelEltDef902 (idef, type, unit, uid,
119:           &                      ifab, iang, ilin, iplas,
120:           &                      nnodes, nodseq, triad, unodes,
121:           &                      nodes, maxdof, nfvars, nivars,
122:           &                      t0,     xg,     xe)
123:
124: elseif (type.eq.910) then
125:
126:          call UelEltDef910 (idef, type, unit, uid,
127:           &                      ifab, iang, ilin, iplas,
128:           &                      nnodes, nodseq, triad, unodes,
129:           &                      nodes, maxdof, nfvars, nivars,
130:           &                      t0,     xg,     xe)
131:
132:  else
133:
134: *        USER-DEFINED ELEMENT TYPE NOT FOUND
135: *        ===================================
136:       write (msg,800) type
137:       call solstp (msg)
138:
139:  endif
140:
141:  100  format(/' <<<< UelEltDef >>>>',
142:       &       /* idef .......: i5
143:       &       /* type .......: i5
144:       &       /* unit .......: i5
145:       &       /* uid .......: i5
146:       &       /* ifab .......: i5
147:       &       /* iang .......: i5
148:       &       /* ilin .......: i5
149:       &       /* iplas .......: i5


User-Defined Elements

User Element Model–Definition Routines

April, 2009

STAGS 5.0 User Manual
The portions that are printed with **boldface** type in this listing of subroutine `UelEltDef` (lines 100–122, inclusive) may be changed, as noted above, to reflect the user’s particular choices in assigning identifier (type) numbers and subroutine names to each of his or her unique User element types. The calling sequences shown here, however, should not be changed.

The observant reader will have noticed that in lines 124–130 of this version of `UelEltDef` there is a test to determine if the current element is a type 910 element, followed by a call to `UelEltDef910` if it is. This logic accommodates STAGS’ “actuator” element, which started out as a developer’s user element and then migrated into current versions of the program as a “regular” or “standard” STAGS element that any analyst can utilize. See the STAGS Elements Manual for more information about this interesting and useful element.

The default version of subroutine `UelEltDef900` (one of the three “children” called by the above-listed version of `UelEltDef`) is shown in the following listing:
**User-Defined Elements**

---

**User Element Model–Definition Routines**

---

**subroutine UelEltDef900**

1: c=deck UelEltDef900
2: c=purpose Perform definition operations for user-written E900
3: c=author -------------------------------
5: c=version Ph: (858)259-2773 or (858)481-9907
6: c=version January 2002
7:
8: #if _usage_
9: *
10: * call UelEltDef900 (idef, type, unit, uid,
11: * ifab, iang, ilin, iplas,
12: * nnodes, nodseq, triad, unodes,
13: * nodes, maxdof, nfvars, nivars,
14: * t0, xgg, xe)
15: *
16: * Input Arguments
17: * ----------------
18: * idef   = current User element definition index
19: * type   = element type (=900)
20: * unit   = current unit number
21: * uid    = user specified element number
22: * ifab   = fabrication code
23: * iang   = material angle flag
24: * ilin   = linearity flag
25: * iplas  = plasticity flag
26: * nnodes = number of nodes
27: * nodseq = nodal sequence for element frame
28: * triad  = material orientation matrix
29: * unodes = element nodes (user-definition)
30: * nodes  = element nodes (STAGS-definition)
31: * nfvars = number of user float variables
32: * nivars = number of user integer variables
33: * t0     = element orientation matrix
34: * xgg    = global nodal coordinates
35: * xe     = local nodal coordinates
36: *
37: #endif
38:
39: ********************************************************************************
40: Subroutine UelEltDef900 (idef, type, unit, uid,
41: & ifab, iang, ilin, iplas,
42: & nnodes, nodseq, triad, unodes,
43: & nodes, maxdof, nfvars, nivars,
44: & t0, xgg, xe)
45: ********************************************************************************
46:
47: # include "keydefs.h"
The two **boldface** lines (73 and 74), here, are to be implemented by the User element developer.

A comprehensive set of FORTRAN- and C-language utility routines is provided with **STAGS** to facilitate the implementation and utilization of user-defined elements in **STAGS’ s1** model-definition processor and in other **STAGS** processors. These utilities are described briefly later in this chapter and are documented more comprehensively in the **STAGS Elements Manual**.

### 13.5 User Element Model–Analysis Routines

When the **STAGS’ s2** processor encounters a User element, it also calls upon a number of user-element “dispatch” routines to perform initialization and computation operations for that **UEL**—
as and if required. Depending on the type of analysis to be performed, \texttt{s2} may call some or all of the user-element dispatch routines that are listed in the following Table:

<table>
<thead>
<tr>
<th>subroutine</th>
<th>raison d’être</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{UelPvDef}</td>
<td>Perform pre-variation operations for User elements</td>
</tr>
<tr>
<td>\texttt{UelMassDef}</td>
<td>Perform mass-computation operations for User elements</td>
</tr>
<tr>
<td>\texttt{UelFIDef}</td>
<td>Compute internal forces for User elements</td>
</tr>
<tr>
<td>\texttt{UelFIDef}</td>
<td>Compute live-loading forces for User elements</td>
</tr>
<tr>
<td>\texttt{UelKmDef}</td>
<td>Compute the material stiffness of the User elements</td>
</tr>
<tr>
<td>\texttt{UelKgDef}</td>
<td>Compute the geometric stiffness of the User elements</td>
</tr>
<tr>
<td>\texttt{UelStrainDef}</td>
<td>Determine the user-element strains</td>
</tr>
<tr>
<td>\texttt{UelStressDef}</td>
<td>Determine the user-element stresses</td>
</tr>
<tr>
<td>\texttt{UelResultDef}</td>
<td>Determine the user-element resultants</td>
</tr>
<tr>
<td>\texttt{UelPrintStrainDef}</td>
<td>Print the user-element strains</td>
</tr>
<tr>
<td>\texttt{UelPrintStressDef}</td>
<td>Print the user-element stresses</td>
</tr>
<tr>
<td>\texttt{UelPrintResultDef}</td>
<td>Print the user-element resultants</td>
</tr>
</tbody>
</table>

Each of these dispatch routines in turn calls a corresponding user-defined routine that is specific to a particular user-defined element type. Subroutine \texttt{UelPvDef}, for example, calls the provided templates \texttt{UelPvDef900}, \texttt{UelPvDef901} and \texttt{UelPvDef902}. Similar template routines are provided for the other dispatch routines. As with the previously-described top-level model-definition routines, the names of these user-element-type-specific model-analysis routines (for hypothetical user types 900, 901 and 902) are arbitrary and may be changed to reflect the user’s particular choice. In any event, it should be obvious that User element name and type choices implemented in \texttt{STAGS’ s1} (model-definition) operations should also be used in \texttt{s2} (model-analysis) operations and in other (post-processing) activities.

For reference purposes, the current default version of the \texttt{UelPvDef} dispatch routine is shown in the following FORTRAN listing:
**subroutine UelPvDef**

1: c=deck UelPvDef
2: c=purpose Perform pre-variation operations for a User element
3: c=author --------------------------------------------------------
5: c=version January 2005
6: #if _usage_
7: *
8: *     call UelPvDef (type, elt, nnodes, active, nvars,
9: *                    nodes, mxdof, nff, ivg, xe)
10: *
11: *
12: *
13: *     Input Arguments
14: *
15: * type = element type: >= 900
16: * elt = element number of type TYPE in the model
17: * nnodes = number of nodes
18: * active = number of active nodes
19: * nvars = number of element variables
20: * nodes = element nodes
21: * mxdof = nodal mxdof vector
22: * nff = nodal DOF-count vector
23: * ivg = nodal DOF vector
24: * xe = nodal coordinates
25: *
26: #endif
27:
28: ************************************************************************
29:          Subroutine UelPvDef (type, elt, nnodes, active, nvars,
30:      &                        nodes, mxdof, nff, ivg, xe)
31: ************************************************************************
32:
33: #     include "keydefs.h"
34:
35: _implicit_none_
36:
37: #     include "stndcm.h"
38:
39: Integer type
40: Integer elt
41: Integer nnodes
42: Integer active
43: Integer nvars
44: Integer nodes(nnodes)
45: Integer mxdof(nnodes)
46: Integer nff(nnodes)
47: Integer ivg(nvars)
subroutine UelPvDef (continued)

48:    _float_    xe(3,nnodes)
49:    Integer    i
50:    Integer    j
51:    Integer    m
52:    Character  msg*48
53:    Logical    NTITLE
54:    
55:    if ( NTITLE( 'X_UelPvDef') )  then
56:      
57:      m = min(nnodes,20)
58:      
59:      write (not,100) type, elt, nnodes, active, nvars
60:      write (not,200) (nodes(i),i=1,m)
61:      write (not,300) (mxdof(i),i=1,m)
62:      write (not,400) (nff  (i),i=1,m)
63:      write (not,500) (ivg  (i),i=1,min(nvars,20))
64:      write (not,600) ((xe(i,j),i=1,3),j=1,m)
65:      
66:      endif
67:      
68: *     PERFORM PRE-VARIATION OPERATIONS FOR A USER-WRITTEN ELEMENT
69: *     ===========================================================
70:      if (type.eq.900) then
71:        
72:        call UelPvDef900 (type, elt, nnodes, active, nvars,
73:                      &                     nodes, mxdof, nff, ivg, xe)
74:        
75:      elseif (type.eq.901) then
76:        
77:        call UelPvDef901 (type, elt, nnodes, active, nvars,
78:                      &                     nodes, mxdof, nff, ivg, xe)
79:        
80:      elseif (type.eq.902) then
81:        
82:        call UelPvDef902 (type, elt, nnodes, active, nvars,
83:                      &                     nodes, mxdof, nff, ivg, xe)
84:        
85:      elseif (type.eq.910) then
86:        
87:      call UelPvDef910 (type, elt, nnodes, active, nvars,
88:                      &                     nodes, mxdof, nff, ivg, xe)
89:        
90:      else
91:        
92: *        USER-WRITTEN ELEMENT TYPE NOT FOUND
93: *        ===================================
94:        write (msg,700) type
95:        call solstp (msg)
The specific UEL type designations and the names of the next–level subroutines that `UelPvDef` calls are typically arbitrary and may be changed to reflect the user’s choices—subject to the architectural constraint that each User element type must have a unique identifier number in the range from 900 to 999, inclusive. The top–level `UelPvDef` dispatch routine documented in this listing treats type 900, 901 and 902 User elements in lines 70–83, inclusive (printed in **boldface** type)—calling next–level subroutines `UelPvDef900`, `UeltPvDef901` and `UelPvDef902` to perform pre–variation operations for the current element when it is necessary to do so. The names of the next–level routines that `UelPvDef` calls are arbitrary, but their calling sequences are fixed and should not be changed.

This version of `UelPvDef` treats a type 910 User element in lines 85–88—calling `UelPvDef910` to do pre–variation work for a type 910 element. If the analyst/developer intends to use the “actuator” element that has been implemented as a type 910 UEL in current versions of **STAGS**, he/she should not touch this portion of the code. If the analyst/developer has implemented something else as a type 910 UEL, anything goes.

In any event, all of the top-level s2-processor dispatch routines and their respective children are documented in Chapter 9 of the **STAGS Elements Manual**.
13.6 User Element Post-Processing Routines

Post-processing capabilities for user-developed elements in STAGS are built primarily around the stapl post-processor, which provides the quickest means for the analyst to visualize his (or her) UEL elements and results obtained with them. User-developed elements can have almost any geometry—simple or complex. Complex user-developed “super” elements can be constructed, for example, from static condensations of one or more groups of standard and/or other types of User elements: these typically “reduce” a complex sub-model to a computationally manageable size by eliminating “interior” nodes and retaining only those nodes that are required to define its internal state—primarily (but not necessarily exclusively)—nodes along the outer boundary of the sub-model. STAGS users can save significant amounts of computer resources with this kind of User element—with some losses in the immediate visibilities of their models and of results obtained with them.

As for the model definition and analysis tasks (in the s1 and s2 processors, respectively), STAGS provides hooks that facilitate the implementation of user-written software for visualizing user-written elements and results obtained with them—in stapl and in other STAGS post-processors, and in s1 (which uses stapl software to generate configuration plots of the analyst’s model).

When any STAGS processor that generates model-configuration and/or -data plots encounters a User element to be visualized, it calls upon one or more of the top-level “dispatch” routines that are listed in the following Table to facilitate the accomplishment of that objective:

<table>
<thead>
<tr>
<th>subroutine</th>
<th>raison d’ être</th>
</tr>
</thead>
<tbody>
<tr>
<td>UelQuadDef</td>
<td>Determine the number of quads to be used to visualize a UEL of type type</td>
</tr>
<tr>
<td>UelGeomDef</td>
<td>Provide nodal connectivity and geometry data for a specific UEL of type type</td>
</tr>
<tr>
<td>UelValuesDef</td>
<td>Provide result value(s) for a specific UEL of type type</td>
</tr>
</tbody>
</table>

Each of these top-level dispatch routines in turn calls a corresponding user-defined routine that is specific to a particular user-defined element type. Subroutine UelQuadDef, for example, calls the provided templates UelQuadDef900, UelQuadDef901 and UelQuadDef902. Similar template routines are provided for the other dispatch routines. As with all of the top-level dispatch routines described here and earlier in this chapter, the names of these user-element-type-specific model-analysis routines (for hypothetical user types 900, 901 and 902) are totally arbitrary and may be changed to reflect the user’s particular choice. For completeness and consistency, User element name and type choices implemented in s1 and s2 should also be implemented in stapl and other (post-processing) processors.
For reference purposes, the current default versions of the **UelQuadDef** and **UelValuesDef** dispatch routines are shown in the following two FORTRAN listings:

```
subroutine UelQuadDef

1: c=deck     UelQuadDef
2: c=purpose  Return # of subelements for visualizing a user-written element
3: c=author   --------------------------------------------------------------
5: c=author   --------------------  Ph: (858)259-2773 or (858)481-9907
6: c=version  October 2004
7:
8: #if _usage_
9: *
10: *     CALL UelQuadDef (type,gelt, nsubs)
11: *
12: *     Input Arguments
13: *     ---------------
14: *     type    =  element type: &ge 900
15: *     gelt    =  global element number
16: *
17: *     Output Arguments
18: *     ---------------
19: *     nsubs   =  number of subelements for visualizing a
20: *     user-written element
21: *
22: #endif
23:
24: ************************************************************************
25:              Subroutine UelQuadDef (type,gelt, nsubs)
26: ************************************************************************
27:
28: #     include   “keydefs.h”
29:
30: _implicit_none_
31:
32: #     include   “stndcm.h”
33:
34: Integer    type
35: Integer    gelt
36: Integer    nsubs
37:
38: Integer    elt
39: Character  msg*48
40: Logical    NTITLE
41:
42: if (NTITLE(‘X_UelQuadDef’) then
43:     write(not,100) type, gelt, elt
44: endif
45:
```

**subroutine UelQuadDef (continued)**

46: * INITIALIZE OUTPUT ARGUMENTS
47: * =============================
48: nsubs = 0
49:
50: * OBTAIN # OF ELEMENT type
51: * ========================
52: call UelEltp (gelt,type, elt)
53:
54: * RETURN NUMBER OF QUADS TO VISUALIZE A USER-WRITTEN ELEMENT
55: * ===============================
56: if (type.eq.900) then
57: 58:      call UelQuadDef900 (type,gelt,elt, nsubs)
59: 60: elseif (type.eq.901) then
61: 62:     call UelQuadDef901 (type,gelt,elt, nsubs)
63: 64: elseif (type.eq.902) then
65: 66:     call UelQuadDef902 (type,gelt,elt, nsubs)
67: 68: elseif (type.eq.910) then
69: 70:      call UelQuadDef910 (type,gelt,elt, nsubs)
71: 72: else
73: 74: * USER-WRITTEN ELEMENT TYPE NOT FOUND
75: * ================================
76:      write (msg,200) type
77:      call solstp (msg)
78: 79: endif
80:
81: format(/' <<<<<<<<<<<<  UelQuadDef  >>>>>>>>>>>>>'
82: &      //' element type ......................', i5
83: &      //' global element number .............', i5
84: &      //' element number of type in model ...', i5)
85: 86: format('UelQuadDef: Undefined user element type: ',i5)
87: 88: end

and
**subroutine UelValuesDef**

1: c=deck  UelValuesDef
2: c=purpose Return solution values for a user-written element
3: c=author -----------------------------------------------
5: c=author ------------------------------------------- Ph: (858)259-2773 or (858)481-9907
6: c=version November 2004
7:
8: #if _usage_
9: *
10: * CALL UelValuesDef (type, gelt, sub, name, loc, dsx,
11: *       step, iplast, layer, zloc, eref,
12: *       nval, values, status)
13: *
14: * Input Arguments
15: *
16: * type    = element type: >=900
17: * gelt    = global element number
18: * sub     = subelement identifier ( 1 <= sub <= Nsubs )
19: *        (cf. subroutine UelQuadDef)
20: * name    = name of data to be returned
21: * loc     = output location: 'N' (nodes) or 'C' (centroid)
22: * dsx     = dataset containing displacement solution
23: * step    = solution step number
24: * iplast  = plasticity flag
25: * layer   = fabrication layer, or 0 (see below)
26: * zloc    = 'B' - Bottom of layer "layer" -- or bottom of
27: *        layer 1, if "layer" = 0; or
28: *        'T' - Top of layer "layer" -- or top of
29: *        layer NLAYS, where NLAYS = total # of
30: *        layers in the fabrication
31: * eref    = 'F' - Fabrication reference frame; or
32: *        'M' - Material reference frame
33: *
34: * Output Arguments
35: *
36: * nval    = if loc = 'N', number of values returned per node
37: *        if loc = 'C', number of values returned
38: * values  = data to be returned:
39: *        loc = 'N' : 4 value sets
40: *        loc = 'C' : 1 value set
41: * status  = success/failure indicator:
42: *        = 0 : data retrieved successfully
43: *        = -1 : data not defined for element
44: *        = -2 : data not defined at requested location
45: *
46: #endif
47:
subroutine UelValuesDef (continued)

Subroutine UelValuesDef (type, gelt, sub, name, loc, dsx, step, iplast, layer, zloc, eref, nval, values, status)

************************************************************************
       Subroutine UelValuesDef (type, gelt, sub, name, loc, dsx, step, iplast, layer, zloc, eref, nval, values, status)
************************************************************************

#     include   "keydefs.h"

#     include   "stndcm.h"
#     include   "dsuni.h"
#     include   "spec.h"
#     include   "unc0.h"
#     include   "unc2.h"
#     include   "xstep.h"

Integer      type
Integer      gelt
Integer      sub
Character    name(*)
Character    loc(*)
Integer      dsx
Integer      step
Integer      iplast
Integer      layer
Character*1  zloc
Character*1  eref
Integer      nval
_Float_      values(*)
Integer      status
Integer      elt
Character    msg*48
Logical      NTITLE
Integer      nwr
Integer      UnitDat

if (NTITLE('X_UelValuesDef')) then
   write(not,100) type, gelt, elt, sub, name, loc, dsx, step
endif

* INITIALIZE STATUS ARGUMENT
* ================
status = 0

* OBTAIN # OF ELEMENT type
* ================
call UelEltp (gelt,type, elt)
subroutine UelValuesDef (continued)

97: * INITIALIZE UNIT OFFSET DATA
98: * ===========================
99: call QROFF (dselt,gelt,1,13,1, nwr,UnitDat)
100: call QREAD (dsunit,UnitDat,3,0, lnod)

103: * RETURN SOLUTION VALUES FOR A USER-WRITTEN ELEMENT
104: * ==============================
105: if (type.eq.900) then
106:  call UelValuesDef900 (type, gelt, elt, sub, name,
107: & loc, dsx, step, xpa, xpb,
108: & iplast, layer, zloc, eref,
109: & nval, values, status)
110: elseif (type.eq.901) then
111:  call UelValuesDef901 (type, gelt, elt, sub, name,
112: & loc, dsx, step, xpa, xpb,
113: & iplast, layer, zloc, eref,
114: & nval, values, status)
115: elseif (type.eq.902) then
116:  call UelValuesDef902 (type, gelt, elt, sub, name,
117: & loc, dsx, step, xpa, xpb,
118: & iplast, layer, zloc, eref,
119: & nval, values, status)
120: elseif (type.eq.910) then
121:  call UelValuesDef910 (type, gelt, elt, sub, name,
122: & loc, dsx, step, xpa, xpb,
123: & iplast, layer, zloc, eref,
124: & nval, values, status)
125: else
126: * USER-WRITTEN ELEMENT TYPE NOT FOUND
127: * ============================
128: write (msg,200) type
129: call solstp (msg)
130: endif
131: format(/' <<<<<<<<<<<  UelValuesDef  >>>>>>>>>>>>>'
132: &'/ element type ......................', i5
133: &'/ global element number .............', i5

138: call solstp (msg)
As noted earlier, the specific User element type designations and the names of the routines that the \texttt{UelQuadDef} and \texttt{UelValuesDef} dispatch routines call—\texttt{UelQuadDef900, UeltQuadDef901} \& \texttt{UelQuadDef902, UelValuesDef900, UeltValuesDef901} \& \texttt{UelValuesDef902} for UEL types 900, 901 and 902, here—are completely arbitrary and may be changed to reflect the user’s particular choices (subject to the architectural constraint that each UEL type must have a unique identifier number in the range from 900 to 999, inclusive). The portions that are printed in \textbf{boldface} type in these listings (lines 56–66 in \texttt{UelQuadDef} and lines 105–124 in \texttt{UelValuesDef}) may be changed to reflect the user’s particular choices in assigning identifier (type) numbers and subroutine names to each unique User element type. The calling sequences shown here should \textit{not} be changed, however.

The three top-level visualization dispatch routines and their respective children are more fully documented in Chapter 9 of the \textit{STAGS Elements Manual}. 

\begin{verbatim}
subroutine UelValuesDef (continued)
145: &   // element number of type in model ..., i5
146: &   // subelement identifier ..........., i5
147: &   // name of requested data ..........., a
148: &   // location of requested data ..........', a
149: &   // solution dataset identifier ........', i5
150: &   // solution step number ..............', i5)
151: 152: 200   format('UelValuesDef: Undefined user element type: ',',i5)
153: 154:   end
\end{verbatim}
13.7 FORTRAN– and C–Language Utility Routines

A comprehensive set of utility routines is provided with STAGS to facilitate the implementation and utilization of user-defined elements in STAGS’ s1 (model–definition) and other processors. Some of these utilities (principally those that are written in the C programming language) are very low-level architectural routines that STAGS users and element developers do not generally need to know about or use. Others (in the FORTRAN language) may be employed effectively in one or more phases of STAGS model–definition, analysis and/or post-processing operations. The principal utilities are described briefly here, and the full set of utilities is documented fully in the STAGS Elements Manual.

Before getting into these descriptions, let us pause briefly to tell the user and element developer that the top-level phase1 routine in STAGS’ s1 processor calls subroutine cards to read all of the input in the user’s INP model–definition input file. Subroutine cards, in turn, calls upon subroutines cread and parse—in cooperation with subroutines XUelParse and parse2—to assist in reading and in parsing user-specified directives—most notably the *userElement and the *userProperty directives. After a STAGS s1 input file has been fully assimilated and processed, subroutine phase1 calls user-element subroutine UelToQdb (which uses a number of lower-level C– and FORTRAN–language routines) to create STAGS databases for all of the specified User element types and variables that are specified in the input file. The average user and element developer should be aware of but not intensely concerned about any of these operations. The user/developer’s attention should be focused exclusively on higher-level operations in defining and utilizing his/her user-developed elements.

Similarly, in the analysis phase of STAGS operations, the top-level work routine in STAGS’ s2 processor also calls upon subroutine cards to read all of the input in the user’s BIN model–analysis input file. In the s2 processor, as in s1, subroutine cards utilizes lower-level subroutines cread and parse to assist in reading and in parsing the user’s input. It is not appropriate for the user to have any *userElement or *userProperty user-element- or user-property-set definition directives here: these directives must be included in the user’s model–definition INP input file. Here, too, the user/developer’s attention should be focused exclusively on higher-level operations in utilizing his/her user-developed elements.

With this in mind, the following Table contains brief descriptions of 19 FORTRAN–language subroutines that the user/developer can employ to load information into and to retrieve information from the STAGS User Element Type databases. The interested user/developer should see Chapter 9 of the STAGS Elements Manual for more information (calling sequences, etc.) about these and other User element type database utility routines.
Table 13.1  User Element Type Database Utility Routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Purpose</th>
<th>Input &amp; Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>UelCount</td>
<td>Retrieve the number of User element types</td>
<td>eltCount = Number of User element types</td>
</tr>
<tr>
<td>UelGetModelCount</td>
<td>Retrieve the number of User elements by type</td>
<td>type = User element type (900 ≤ type ≤ 999)</td>
</tr>
<tr>
<td></td>
<td>count = Number of User elements of type type in the model</td>
<td>count = Number of User elements of type type in the model</td>
</tr>
<tr>
<td></td>
<td>dse9xx = User element dataset handle for type type</td>
<td>dse9xx = User element dataset handle for type type</td>
</tr>
<tr>
<td>UelGetUnitCount</td>
<td>Retrieve the number of User elements by type and unit</td>
<td>type = User element type (900 ≤ type ≤ 999)</td>
</tr>
<tr>
<td></td>
<td>unit = STAGS model unit identifier</td>
<td>unit = STAGS model unit identifier</td>
</tr>
<tr>
<td></td>
<td>count = Number of User elements of type type in unit unit</td>
<td>count = Number of User elements of type type in unit unit</td>
</tr>
<tr>
<td></td>
<td>dse9xx = User element dataset handle for type type</td>
<td>dse9xx = User element dataset handle for type type</td>
</tr>
<tr>
<td>UelIndex</td>
<td>Retrieve the index for a User element type</td>
<td>type = User element type (900 ≤ type ≤ 999)</td>
</tr>
<tr>
<td></td>
<td>elt = Index for User element of type type (if the User element type</td>
<td>elt = Index for User element of type type (if the User element type does not exist, then zero is</td>
</tr>
<tr>
<td></td>
<td>does not exist, then zero is returned for the element type index)</td>
<td>returned for the element type index)</td>
</tr>
<tr>
<td>UelInfo</td>
<td>Retrieve metadata for a User element type</td>
<td>elt = Index of User element type</td>
</tr>
<tr>
<td></td>
<td>nodeCount = Number of nodes for User element type type</td>
<td>nodeCount = Number of nodes for User element type type</td>
</tr>
<tr>
<td></td>
<td>nevars = Number of element variables</td>
<td>nevars = Number of element variables</td>
</tr>
<tr>
<td></td>
<td>nodeSeq = Node sequence (3 or 4) to compute element frame</td>
<td>nodeSeq = Node sequence (3 or 4) to compute element frame</td>
</tr>
<tr>
<td></td>
<td>nfvars = Number of element type float variables</td>
<td>nfvars = Number of element type float variables</td>
</tr>
<tr>
<td></td>
<td>nivars = Number of element type integer variables</td>
<td>nivars = Number of element type integer variables</td>
</tr>
<tr>
<td></td>
<td>name = Name of User element type</td>
<td>name = Name of User element type</td>
</tr>
<tr>
<td></td>
<td>nameLen = Number of characters in name</td>
<td>nameLen = Number of characters in name</td>
</tr>
<tr>
<td>UelVarInfo</td>
<td>Retrieve the number of User element type variables</td>
<td>elt = Index of User element type</td>
</tr>
<tr>
<td></td>
<td>floCount = Number of float variables for this User element type</td>
<td>floCount = Number of float variables for this User element type</td>
</tr>
<tr>
<td></td>
<td>intCount = Number of integer variables for this User element type</td>
<td>intCount = Number of integer variables for this User element type</td>
</tr>
<tr>
<td>UelFloVar</td>
<td>Retrieve metadata for a User element type float variable</td>
<td>elt = Index of User element type</td>
</tr>
<tr>
<td></td>
<td>var = Index of User element type float variable</td>
<td>var = Index of User element type float variable</td>
</tr>
<tr>
<td></td>
<td>size = Variable size (in logical words)</td>
<td>size = Variable size (in logical words)</td>
</tr>
<tr>
<td></td>
<td>histIdx = Variable index in history buffer (if positive)</td>
<td>histIdx = Variable index in history buffer (if positive)</td>
</tr>
<tr>
<td></td>
<td>name = Variable name</td>
<td>name = Variable name</td>
</tr>
<tr>
<td></td>
<td>nameLen = Length of variable name</td>
<td>nameLen = Length of variable name</td>
</tr>
<tr>
<td>UelIntVar</td>
<td>Retrieve metadata for a User element type integer variable</td>
<td>elt = Index of User element type</td>
</tr>
<tr>
<td></td>
<td>var = Index of User element type integer variable</td>
<td>var = Index of User element type integer variable</td>
</tr>
<tr>
<td></td>
<td>size = Variable size in logical words</td>
<td>size = Variable size in logical words</td>
</tr>
<tr>
<td></td>
<td>histIdx = Variable index in history buffer (if positive)</td>
<td>histIdx = Variable index in history buffer (if positive)</td>
</tr>
<tr>
<td></td>
<td>name = Variable name</td>
<td>name = Variable name</td>
</tr>
<tr>
<td></td>
<td>nameLen = Length of variable name</td>
<td>nameLen = Length of variable name</td>
</tr>
</tbody>
</table>
### Table 13.1 User Element Type Database Utility Routines (continued)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Purpose</th>
<th>Input &amp; Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UelSetFloVar</strong></td>
<td>Set User element type float variable(s)</td>
<td><code>type = User element type (900 ≤ type ≤ 999)</code>&lt;br&gt;<code>elt = Index of User element type</code>&lt;br&gt;<code>name = Name of float variable</code>&lt;br&gt;<code>data = Float variable data</code>&lt;br&gt;<code>size = Size of data stored (in logical words)</code></td>
</tr>
<tr>
<td><strong>UelGetFloVar</strong></td>
<td>Retrieve a User element type float variable(s); it is the caller’s responsibility to ensure that the data buffer is large enough to hold the returned items</td>
<td><code>type = User element type (900 ≤ type ≤ 999)</code>&lt;br&gt;<code>elt = Index of User element type</code>&lt;br&gt;<code>name = Name of float variable (‘*’ =&gt; all float variables)</code>&lt;br&gt;<code>data = Float variable data vector</code>&lt;br&gt;<code>size = Size of data (in logical words); set to zero if name does not exist</code></td>
</tr>
<tr>
<td><strong>UelSetIntVar</strong></td>
<td>Set User element type integer variable(s)</td>
<td><code>type = User element type (900 ≤ type ≤ 999)</code>&lt;br&gt;<code>elt = Index of User element type</code>&lt;br&gt;<code>name = Name of integer variable (‘*’ =&gt; all integer variables)</code>&lt;br&gt;<code>data = Integer variable data</code>&lt;br&gt;<code>size = Size of data stored (in logical words)</code></td>
</tr>
<tr>
<td><strong>UelGetIntVar</strong></td>
<td>Retrieve User element type integer variable(s); it is the caller’s responsibility to ensure that the data buffer is large enough to hold the returned items</td>
<td><code>type = User element type (900 ≤ type ≤ 999)</code>&lt;br&gt;<code>elt = Index of User element type</code>&lt;br&gt;<code>name = Name of integer variable (‘*’ =&gt; all integer variables)</code>&lt;br&gt;<code>data = Integer variable data vector</code>&lt;br&gt;<code>size = Size of data (in logical words); </code>size<code> is set to zero if name does not exist</code></td>
</tr>
<tr>
<td><strong>UelSetHistVar</strong></td>
<td>Set all of the User element historical variables</td>
<td><code>type = User element type (900 ≤ type ≤ 999)</code>&lt;br&gt;<code>elt = Index of User element type</code>&lt;br&gt;<code>data = history variable</code>&lt;br&gt;<code>size = Size of data stored (in floating point words)</code></td>
</tr>
<tr>
<td><strong>UelSetHistFloVar</strong></td>
<td>Set a User element historical floating point variable</td>
<td><code>type = User element type (900 ≤ type ≤ 999)</code>&lt;br&gt;<code>elt = Index of User element type</code>&lt;br&gt;<code>name = Name of float variable (‘*’ =&gt; all float variables)</code>&lt;br&gt;<code>data = floating point history variable(s)</code>&lt;br&gt;<code>size = Size of data stored (in logical words)</code></td>
</tr>
<tr>
<td><strong>UelSetHistIntVar</strong></td>
<td>Set a User element historical integer variable</td>
<td><code>type = User element type (900 ≤ type ≤ 999)</code>&lt;br&gt;<code>elt = Index of User element type</code>&lt;br&gt;<code>name = Name of integer variable (‘*’ =&gt; all integer variables)</code>&lt;br&gt;<code>data = Integer history variable(s)</code>&lt;br&gt;<code>size = Size of data stored (in logical words)</code></td>
</tr>
</tbody>
</table>
The following Table contains brief descriptions of 9 FORTRAN–language subroutines that the user/developer can employ to load information into and to retrieve information from the STAGS User property set databases. The interested user/developer should see Chapter 9 of the STAGS Elements Manual for more information (calling sequences, etc.) about these and other User property set database utility routines.

<table>
<thead>
<tr>
<th>Routine</th>
<th>Purpose</th>
<th>Input &amp; Output</th>
</tr>
</thead>
</table>
| UelGetHistFloVar | Retrieve a User element historical floating point variable | type = User element type (900 ≤ type ≤ 999)  
elt = Index of User element type  
name = Name of float variable ("*" => all float variables)  
data = floating point history variable(s)  
size = Size of data retrieved (in logical words) |
| UelSetHistIntVar | Retrieve a User element historical integer variable | type = User element type (900 ≤ type ≤ 999)  
elt = Index of User element type  
name = Name of integer variable ("*" => all integer variables)  
data = Integer history variable(s)  
size = Size of data retrieved (in logical words) |
| UelSetStr        | Map a character string into an array of encoded integers | string = Character string to be copied into ibuff (trailing spaces are ignored).  
nlen = ibuff length holding string  
ibuff = Integer array with the structure: {stringLen, char1, char2, ..., charN} |
| UelGetStr        | Retrieve a character string from an array of integer-encoded characters | ibuff = Integer array with the structure: {stringLen, char1, char2, ..., charN}  
stringLen = Length of character string string  
string = Character string retrieved from ibuff |
## Table 13.2 User Property Set Database Utility Routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Purpose</th>
<th>Input &amp; Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPropCount</td>
<td>Retrieve the number of User property sets</td>
<td>propCount = Number of User property sets</td>
</tr>
</tbody>
</table>
| UPropIndex     | Retrieve the index for a user property set        | id = User property set identifier (specified by the user)  
prop = Index of User property set                                                                 |
| UPropInfo      | Retrieve metadata for a user property item        | prop = User property set index  
id = User property set identifier (specified by the user)  
floCount = Number of user property set float items  
intCount = Number of user property set integer items  
name = Name of user property set  
nameLen = Number of characters in name                                                                 |
| UPropFloVal    | Retrieve metadata for a user property set float value | prop = User property set index  
item = User property set float item index  
data = Float property data  
name = Property item name  
nameLen = Property item name length                                                                 |
| UelSetFloProp  | Set user property float item(s)                   | id = User property set identifier (specified by the user)  
name = Name of float property ('*' => all float properties)  
data = float property data items  
size = Size of data (in logical words); set to zero if name does not exist |
| UelGetFloProp  | Retrieve float user property item(s)              | id = User property set identifier (specified by the user)  
name = Name of float property ('*' => all float properties)  
data = Float property data items (it is the caller's responsibility to ensure that the data buffer is large enough to hold the returned items)  
size = Size of data (in logical words); set to zero if name does not exist |
| UPropIntVal    | Retrieve metadata for a user property set integer value | prop = User property set index  
item = User property set integer item index  
data = Integer property data  
name = Property item name  
nameLen = Property item name length                                                                 |
| UelSetIntProp  | Set user property integer item(s)                 | id = User property set identifier (specified by the user)  
name = Name of integer property ('*' => all integer properties)  
data = integer property data items  
size = Size of data (in logical words); set to zero if name does not exist |
| UelGetIntProp  | Retrieve integer user property item(s); it is the caller's responsibility to ensure that the data buffer is large enough to hold the returned items | id = User property set identifier (specified by the user)  
name = Name of integer property ('*' => all integer properties)  
data = Integer property data items  
size = Size of data (in logical words); set to zero if name does not exist |
Last, but not least, the following Table contains brief descriptions of 4 FORTRAN-language routines that the analyst/developer can employ to retrieve element-property information and results from STAGS databases when STAGS’ GCP processor is employed in specifying material properties and element fabrications. See Chapter 9 of the STAGS Elements Manual for more information about these and other GCP-interface routines.

### Table 13.3 User Element GCP-Interface Routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Purpose</th>
<th>Input &amp; Output</th>
</tr>
</thead>
</table>
| e9xxcp  | Retrieve the CC matrix for a UEL-GCP installation | nlin = Nonlinear-strain/displacement-relations flag
iplast = Plasticity-theory use flag
il = Local-angles-defining-planar-rotation flag
ag = Local angles defining planar rotations
ccp = Full (non-triangular) ABD matrix
status = Success/change-step-size/failure flag |
| e9xxgss | Obtain the stress resultants for a UEL-GCP installation | nlin = Nonlinear-strain/displacement-relations flag
iplast = Plasticity-theory use flag
pa = Load factor for system A
pb = Load factor for system B
kinflg = kinematic type flag for the element
il = Local-angles-defining-planar-rotation flag
ag = Local angles defining planar rotations
strainr = Element strains/curvatures
stressr = Stress resultants corresponding to given strains
edens = Energy density (nips values)
status = Success/change-step-size/failure flag |
| e9xxms  | Obtain the inertia terms for a UEL-GCP installation | inertia = Mass per unit volume, surface or length |
| e9xxpse | Obtain the stress resultants for a multi-layer UEL-GCP installation | step = Load step number
pa = Load factor for system A
pb = Load factor for system B
il = Local-angles-defining-planar-rotation flag
ag = Local angles defining planar rotations (nips values)
iplast = Plasticity-theory use flag
ilayr = Layer # for which output is desired
zloc = Z location (through thickness) where output is desired
eref = Reference frame for stress/strain computations
zet = Angle between reference frame and direction of results
strainr = Element strains/curvatures (nsr*nips points)
epsilon = Strain vector (nsc components at nsp points)
plep = Plastic strains (nsc components at nsp points)
nsc = Number of strain components
epseff = Effective strain at nsp points
sigma = Stress vector (nsc components at nsp points)
sigeff = Effective stress at nsp points
status = Success/change-step-size/failure flag |
13.8 Uniform Beam Example

This section shows the use of some of STAGS’ User element features in solving the textbook problem that is shown in Figure 13.1:

![Figure 13.1 Uniform Beam Example Problem](image)

This straight, uniform beam with a circular cross section—10.0 inches long with a 1.0 inch radius—is fully clamped at $x=0$ (End A) and is loaded at the free end at $x=L$ (End B) with 1.0 pound forces in the $y$– and $z$–directions.

The user-defined beam element that is to be used in solving this problem is described by Przemieniecki.* It has uniform cross-section and material properties and has two active nodes, each with six degrees of freedom. A third (inactive) node is used for reference purposes: with the two active nodes, the third node defines the beam’s principle $x$–$y$ plane. The nodal force directions for this user-defined beam element are shown in Figure 13.2.

The nonzero stresses at node A for this element are

\[
\sigma_x = -\frac{F_1}{A} - \frac{F_5 c_z}{I_y} + \frac{F_6 c_y}{I_z} \quad (13.1)
\]

\[
\tau_{xy} = SF_T \frac{F_4 c_z}{J} \quad (13.2)
\]

\[
\tau_{xz} = -SF_T \frac{F_4 c_y}{J} \quad (13.3)
\]

where A is the area of the cross-section of the beam \((= \pi R^2)\), where \((SF_T = 1)\), and where \(c_y\) and \(c_z\) are defined in Figure 13.2. The \(c_y\) and \(c_z\) parameters define sampling point coordinates on the beam cross section, for stress recovery. The nonzero stresses at node B for this element are

\[
\sigma_x = \frac{F_7}{A} + \frac{F_{11} c_z}{I_y} - \frac{F_{12} c_y}{I_z} \quad (13.4)
\]
The STAGS problem description file (ubeam1.inp) is given below (with line numbers on the left shown for reference purposes):

```
ubeam1.inp  input file for uniform beam example

1:  Straight Uniform Beam via User Defined Elements
2:  0  0  0  0  0  0  0                                $ b1
3:  0  1  0  0  0  0  0  0  0  0                       $ b2
4:  0  0  0  0  0  0                                  $ b3
5:  0  0  0  0  0  0                                  $ h1
6:  $
7:  $====================================================
8:  $   User Element Definitions
9:  $====================================================
10:  $
11:  *userElement name="Uniform Beam Element"  type=900  nodes=3
12:  $
13:  *dofOrdering
14:  $
15:  $    Node  DOF...
16:  $    --------------
17:   1     1 2 3 4 5 6
18:   2     1 2 3 4 5 6
19:   3     0
20:  $
21:  *nodeSequence
22:  $
23:  $    Nodes...
24:  $    -------
25:   1 2 3
26:  $
27:  *floatVariables
28:  $
29:  $    Name       Size
30:  $    --------------
31:   Area       1
```

\[
\tau_{xy} = -SF_T \frac{F_{10} c_z}{J} \quad \text{(13.5)}
\]

\[
\tau_{xz} = SF_T \frac{F_{10} c_y}{J} \quad \text{(13.6)}
\]
ubeam1.inp  input file for uniform beam example (continued)

32:  AreaInShearY    1
33:  AreaInShearZ    1
34:  Iy              1
35:  Iz              1
36:  J               1
37:  Length          1
38:  Material        1
39:  MaxCompression  1
40:  MaxShear        1
41:  MaxTension      1
42:  ShearDefY       1
43:  ShearDefZ       1
44:  ShearFactorT    1
45:  ShearFactorY    1
46:  ShearFactorZ    1
47:
48:  *end userElement
49:
50:  *userProperty  name="Standard Data -- Uniform Beam Element"  id=900
51:
52:  *integerProps
53:     $  Required Standard Data
54:     $  ----------------------
55:     ActiveNodes     2
56:     SamplingCount  10
57:     StrainCount     6
58:     StressCount     6
59:  *floatProps
60:     $  Nodal Stress/Strain Sampling Points
61:     $  -------------------------------
62:     y1  0.0
63:     z1  1.0
64:
65:     y2  1.0
66:     z2  0.0
67:
68:     y3  0.0
69:     z3 -1.0
70:
71:     y4 -1.0
72:     z4  0.0
73:
74:     y5  0.0
75:     z5  0.0
76:
77:  *end userProperty
78:
The observant reader will note that line 11 in this model-definition input file is the starting point for a *userElement directive defining a User element type which will (arbitrarily) be identified as a type 900 element. Each type 900 element is defined by three nodes whose nodal
DOF are specified with the *dofOrdering directive, as shown in lines 17–19. The beam element’s active nodes each have six DOF while the reference node has zero DOF.

Note: Each node in a user-defined element must currently have 0, 3 or 6 DOF.

The *nodeSequence directive (on lines 21–26) must contain either three or four indices identifying nodes in the previous node-DOF specification that are to be used to define the local reference frame for the User element (see Section 13.2 for a discussion of how an element’s local frame is constructed). In this case, the sequence 1,2,3 specifies that the first, second and third node in the previous node-DOF list are to be used to define the local reference frame for the element.

Each type 900 beam element has a set of sixteen floating-point data items as specified by the *floatVariables directive on lines 27–47. These data items specify an element’s characteristic geometry data (Area, AreaInShear, etc.) and solution results (MaxCompression, MaxShear, etc.). Note that the variable called Material is used as a pointer to the element’s material data that are specified later as User property information. There are no integer data for this User element type, so the *userElement directive terminates at line 48.

In this example, the User property items that are required for the User element type definition (ActiveNodes, SamplingCount, StrainCount and StressCount) are combined with additional data using the *userProperty directive beginning on line 52. Note that this user property set has an identifier value that is identical to the identifier of the User element type definition (i.e., 900); this causes this particular property set to be associated with User element type definition identified by 900. In addition to the required integer element data given on lines 56–58, this property set also contains floating-point data (in lines 59–76)—these data define y–z sampling point coordinates where stresses and strains are to be evaluated at each end of a beam element, as shown in the following Figure:

![Diagram](https://via.placeholder.com/150)

This first property set definition is terminated at line 77. A second user property set directive, contained on lines 79–92, defines material data for aluminum 6061-T6. Full use of long property names is used here to make the items self-describing. Note that while any unique identifier could be used to tag this property set (other than 900, as discussed above), an id value of 6061 seems appropriate. This property set terminates on line 92.
Following the one User element type definition and two property set definitions, standard STAGS input continues at line 97 with the definition of an element unit containing 11 nodes spaced one inch along the x–axis beginning at the global origin. These nodes are unconstrained in both translation and rotation. A twelfth node is located at (5,1,0); this node is fully fixed and is used as the principle x–y plane reference node for each of the beam elements.

Line 102 tells STAGS that one user-defined element type is used in this model. Lines 103–112 define ten type 900 elements and their variable data. The first beam element is connected between nodes 1 and 2 and uses node 12 as its principle x–y plane reference node. Subsequent elements are connected between nodes (2,3), (3,4), etc. All elements use node 12 as their principle x–y plane reference node.

Values for the element’s variable data (which were declared on lines 31–46) are contained on lines 106–111. In general, all User element floating-point data are specified first, followed by all integer data. These data are entered in the order that they were declared in *floatVariables and *integerVariables directives within the *userElement directive. Regardless of how many *floatVariables and *integerVariables directives are used, or in what order they are actually given, float and integer variable declarations are separately combined in the order that each type is declared. Note that the material type identifier (Material) is defined as a floating-point variable in the *floatVariables part of the *userElement definition (on line 38) for this User element, so the value of Material is specified as a floating-point number (6061.0) on line 108 of the user’s input file. If Material had been defined as an integer variable (in the *integerVariables part of the *userElement definition, its value would have to have been specified later, as an integer number.

As noted earlier, some of these User element variables characterize the element (and require specified input values) while the other variables will be specified later with values that are to be filled-in by user-written code. For example, the data on line 106 and 107 give the beam cross sectional area, effective area in shear, area moments of inertia and torsion factor. Data values on lines 108 and 109 are all place holders, with the exception of the value for Material, and will be specified later by user-written code. The value of Material being set to 6061 is used by the analyst to link these elements to the aluminum material data in the property set identified as 6061. Data on lines 110 and 111 are mixed; the shear deformation parameters (ShearDefY and ShearDefZ) are computed by code and the shear factors (ShearFactorT, ShearFactorY and ShearFactorZ) are specified.

Standard STAGS input continues on lines 116–122, which specify the boundary conditions and applied loads. On line 118, all displacement and rotation DOF for the root node are specified to
have zero values. Lines 119–120 specify that one pound force loadings are applied at node 11 in the y– and z–directions, respectively.

Finally, line 122 requests that STAGS output displacement and internal force solution results.

User Element Definition Routines

This example problem does not require any special processing in STAGS s1 processor for the type 900 user-defined beam element. The user-written routines called by the dispatchers UelEltDef, UelLoadDef and UelPressDef (cf. Section 13.4) only need to be implemented as empty routines. The template routines UelEltDef900, UelLoadDef900 and UelPressDef900 that are supplied as part of the STAGS distribution will work perfectly fine, for this problem. Because these template routines are automatically linked as part of standard STAGS, no additional actions are necessary.

Since this user-defined element will be used in a static nonlinear analysis, only a few model-analysis routines will need to be written to support STAGS’ s2 processing (cf. Section 13.5). Recall that STAGS’ s2 processor typically calls twelve top-level dispatch routines during the solution process:

- UelPvDef,
- UelMassDef,
- UelFiDef,
- UelFiDef900,
- UelKmDef,
- UelKgDef,
- UelStrainDef,
- UelStressDef,
- UelResultantDef,
- UelPrintStrainDef
- UelPrintStressDef
- UelPrintResultantDef

and

- UelPvDef900,
- UelFiDef900,
- UelKmDef900,

Each of these dispatch routines in turn calls a corresponding user-written subroutine that is specific to each user-defined element type. For this example, the following user-written routines should typically be supplied for this type 900 element:
UelStrainDef900,
UelStressDef900,
UelPrintStrainDef900

and

UelPrintStressDef900.

The UelMassDef900, UelFlDef900 and UelKgDef900 subroutines only need to be implemented as empty routines. The template versions of UelMassDef900, UelFlDef900 and UelKgDef900 that are supplied as part of the STAGS distribution will work fine, for this problem, so no additional work on them is required.
User beam pre-variation definition routine—UelPvDef900

The purpose of a User element pre-variation definition routine (such as `UelPvDef900`) is to compute and store element parameters using operations that only need to be performed once. Each User element pre-variation routine that is called by the `UelPvDef` dispatcher is provided for this purpose. The pre-variation definition routine for this User element is presented below (with line numbers shown on the left for reference purposes):

```
subroutine UelPvDef900
  c=deck     UelPvDef900
  c=purpose  Perform pre-variation operations for User element E900
  c=author   --------------------------------------------------------
  c=author   Lyle W. Swenson, Jr.  Knowledge Management Systems, Inc.
  c=author   --------------------  Ph: (858)259-2773 or (858)481-9907
  c=version  December 2001
  #if _usage_
  *     call UelPvDef900 (type, elt, nnodes, active, nvars,
  *                       nodes, mxdof, nff, ivg, xe)
  *     Input Arguments
  *     ---------------
  *     type    =  element type (=900)
  *     elt     =  element number of type TYPE in the model
  *     nnodes  =  number of nodes
  *     active  =  number of active nodes
  *     nvars   =  number of element variables
  *     nodes   =  element nodes
  *     mxdof   =  nodal maxdof vector
  *     nff     =  nodal DOF-count vector
  *     ivg     =  nodal DOF vector
  *     xe      =  nodal coordinates
  #endif
  #     include   "keydefs.h"
  _implicit_none_
  #     include   "stndcm.h"
```

13-42  April, 2009  STAGS 5.0 User Manual
Uniform Beam Example

User–Defined Elements

40: Integer elt
41: Integer nnodes
42: Integer active
43: Integer nvars
44: Integer nodes(nnodes)
45: Integer mxdof(nnodes)
46: Integer nff(nnodes)
47: Integer ivg(nvars)
48: _float_ xe(3,nnodes)
49: 
50: * TO BE IMPLEMENTED BY THE USER
51: * ############################################################
52: 
53: Integer size
54: 
55: _float_ Asy
56: _float_ Asz
57: _float_ E
58: _float_ G
59: _float_ Iy
60: _float_ Iz
61: _float_ L
62: _float_ Mat
63: _float_ SDy
64: _float_ SDz
65: 
66: call UelGetFloVar(type,elt,'AreaInShearY', Asy, size)
67: call UelGetFloVar(type,elt,'AreaInShearZ', Asz, size)
68: call UelGetFloVar(type,elt,'Iy', Iy, size)
69: call UelGetFloVar(type,elt,'Iz', Iz, size)
70: call UelGetFloVar(type,elt,'Material', Mat, size)
71: 
72: call UelGetFloProp(nint(Mat),'E', E, size)
73: call UelGetFloProp(nint(Mat),'G', G, size)
74: 
75: * Compute Beam Length and Shear Deformation Parameters
76: * ################################################################################
77: L = sqrt( (xe(1,1)-xe(1,2))**2 +
78: & (xe(2,1)-xe(2,2))**2 +
79: & (xe(3,1)-xe(3,2))**2 )
80: 
81: SDy = 12*E*Iz/(G*Asy*L*L)
82: SDz = 12*E*Iy/(G*Asz*L*L)
83: 
84: call UelSetFloVar(type,elt,'Length', L, size)
85: call UelSetFloVar(type,elt,'ShearDefY', SDy, size)
86: call UelSetFloVar(type,elt,'ShearDefZ', SDz, size)
87: 
88: end
Lines 1–52 are provided as part of the standard UelPvDef900 template routine. The idea of any template is to provide developers all the data they might need to calculate the requested output or internal intermediate data.

Lines 53–64 define all the variables that are needed for the local operations. For demonstration purposes, this routine computes some of the beam parameters that will be used later by other type 900 routines.

Lines 66–70 retrieve some floating-point User element variables for a specific type 900 element. Notice that the names in the UelGetFloVar calls were created by the user in the *userElement directive in the model description file. The material identification parameter (Material) is used as a User property set identifier in lines 72–73 to retrieve the material properties for this particular element.

Computations on lines 77–82 are performed to compute the beam element’s length and shear deformation parameters. These computed parameters are stored as part of the User element variables on lines 84–86. While these computations are very simple, they do demonstrate the purpose of a User element prevariation routine—to compute invariant element parameters once and store them for later use.
User beam internal force vector definition routine—UelFiDef900

The purpose of a User element internal force definition routine (such as UelFiDef900) is to compute and return the internal force vector, a plastic strain code and the internal energy for a specific UEL. Each User element internal force definition routine that is called by the UelFiDef dispatcher is provided for this purpose. The internal force definition routine for this User element is presented below (with line numbers shown on the left for reference purposes):

```fortran
subroutine UelFiDef900
  c=deck     UelFiDef900
  c=purpose  Compute internal force vector for user-written E900
  c=author   --------------------------------------------------------
  c=author   Lyle W. Swenson, Jr.  Knowledge Management Systems, Inc.
  c=author   --------------------  Ph: (858)259-2773 or (858)481-9907
  c=version  December 2001
  
  #if _usage_
  *
  *     call UelFiDef900 (type, elt, nlin, iplast, istep,
  *                       pa, pb, active, nvars, ix,
  *                       de, fi, iplmax, energy)
  *
  *     Input Arguments
  *     ----------------
  *     type    =  element type (=900)
  *     elt     =  element number of type TYPE in the model
  *     nlin    =  =0 - do not use nonlinear strain/displ. relations
  *                =1 - use nonlinear relations for applicable elements
  *     iplast  =  =0 - do not use plasticity theory
  *                >0 - plastic strains excessive, stop first variation
  *     istep   =  step number of displacement vector
  *     pa      =  system A load factor
  *     pb      =  system B load factor
  *     active  =  number of active element nodes
  *     nvars   =  number of element variables
  *     ix      =  node permutation array
  *     de      =  element local deformation vector
  *     
  *     Output Arguments
  *     ----------------
  *     fi      =  internal forces
  *     iplmax  =  =0 - plastic strains not excessive (plasticity only)
  *                >0 - plastic strains excessive, stop first variation
  *     energy  =  element internal energy
  *
  #endif

************************************************************************
```

---

STAGS 5.0 User Manual  
April, 2009  
13-45
Subroutine UelFiDef900 (type, elt, nlin, iplast, istep,
& pa, pb, active, nvars, ix,
& de, fi, iplmax, energy)
************************************************************************

#include "keydefs.h"

_implicit_none_

#include "stndcm.h"

Integer type
Integer elt
Integer nlin
Integer iplast
Integer istep
_Float_ pa
_Float_ pb
Integer active
Integer nvars
Integer ix(active)
_Float_ de(nvars)
_Float_ fi(nvars)
Integer iplmax
_Float_ energy

* TO BE IMPLEMENTED BY THE USER
*

Integer i
Integer size
_Float_ A
_Float_ E
_Float_ G
_Float_ Iy
_Float_ Iz
_Float_ J
_Float_ L
_Float_ Mat
_Float_ SDy
_Float_ SDz
_Float_ K0101
_Float_ K0202
_Float_ K0206
_Float_ K0303
_Float_ K0305
_Float_ K0404
_Float_ K0505
_Float_ K0511
_Float_ K0606
Uniform Beam Example

Retrieve Element And Material Properties

call UelGetFloVar (type,elt,'Area', A, size)
call UelGetFloVar (type,elt,'Iy', Iy, size)
call UelGetFloVar (type,elt,'Iz', Iz, size)
call UelGetFloVar (type,elt,'J', J, size)
call UelGetFloVar (type,elt,'Length', L, size)
call UelGetFloVar (type,elt,'Material', Mat, size)
call UelGetFloVar (type,elt,'ShearDefY', SDy, size)
call UelGetFloVar (type,elt,'ShearDefZ', SDz, size)
call UelGetFloProp (nint(Mat),'E', E, size)
call UelGetFloProp (nint(Mat),'G', G, size)

Stiffness Coefficients

K0101 = E*A/L
K0202 = 12*E*Iz/((1+SDy)*L**3)
K0206 =  6*E*Iz/((1+SDy)*L**2)
K0303 = 12*E*Iy/((1+SDz)*L**3)
K0305 =  6*E*Iy/((1+SDz)*L**2)
K0404 = G*J/L
K0505 = (4+SDz)*E*Iy/((1+SDz)*L)
K0511 = (2-SDz)*E*Iy/((1+SDz)*L)
K0606 = (4+SDy)*E*Iz/((1+SDy)*L)
K0612 = (2-SDy)*E*Iz/((1+SDy)*L)

X-Forces: S1 and S7
fi(1) = K0101 * (de(1)-de(7))
fi(7) = -fi(1)

Y-Forces: S2 and S8
fi(2) = K0202 * (de(2)-de(8)) + K0206 * (de(6)+de(12))
fi(8) = -fi(2)

Z-Forces: S3 and S9
fi(3) = K0303 * (de(3)-de(9)) - K0305 * (de(5)+de(11))
fi(9) = -fi(3)

X-Moments: S4 and S10
fi( 4) = K0404 * (de(4) - de(10))
User-Defined Elements

Uniform Beam Example

144: \( \text{fi}(10) = -\text{fi}(4) \)
145:
146: * Y-Moments: S5 and S11
147: * ---------------------
148: \( \text{fi}(5) = K0505 \times \text{de}(5) + K0511 \times \text{de}(11) + K0305 \times (\text{de}(9)-\text{de}(3)) \)
149: \( \text{fi}(11) = K0511 \times \text{de}(5) + K0505 \times \text{de}(11) + K0305 \times (\text{de}(9)-\text{de}(3)) \)
150:
151: * Z-Moments: S6 and S12
152: * ---------------------
153: \( \text{fi}(6) = K0606 \times \text{de}(6) + K0612 \times \text{de}(12) + K0206 \times (\text{de}(2)-\text{de}(8)) \)
154: \( \text{fi}(12) = K0612 \times \text{de}(6) + K0606 \times \text{de}(12) + K0206 \times (\text{de}(2)-\text{de}(8)) \)
155:
156: * Plasticity Code
157: * ------------------
158: iplmax = 0
159:
160: * Element Energy
161: * ------------------
162: energy = 0.d0
163:
164: do 10 i=1,nvars
165:     energy = energy + de(i)*fi(i)
166: 10    continue
167:
168: energy = 0.5d0 * energy
169:
170: end

Lines 1–68 are provided as part of the standard **UelFiDef900** template routine. The idea of any template is to provide developers all the data they might need to calculate the requested output or internal intermediate data.

Lines 69–92 define all the variables that are needed for the local operations. This routine is going to compute an element’s internal force vector. Lines 96–103 retrieve some floating-point User element variables for a specific type 900 element. Notice that the names in the **UelGetFloVar** calls were created by the user in the *userElement* directive in the model description file. The material identification parameter (Material) is utilized as a User property set identifier in lines 105–106 to retrieve the material properties for this particular element.

Computations on lines 110–124 are performed to calculate the beam element’s stiffness coefficients. These stiffness values are then used on lines 128–154, along with the element’s deformations, to determine the element’s internal force components. Finally, the plastic strain code is set to zero and the element’s internal energy are computed on lines 158 and 162–168, respectively. While these computations are very simple, they do demonstrate the general approach for any User element internal force definition routine.
User beam material stiffness matrix definition routine—UelKmDef900

The purpose of a User element material stiffness matrix definition routine (such as UelKmDef900) is to compute and return the material stiffness matrix for a specific UEL. Each User element material stiffness matrix definition routine called by the UelKmDef dispatcher is provided for this purpose. The material stiffness matrix definition routine for this User element is presented below (with line numbers shown on the left for reference purposes):

```fortran
subroutine UelKmDef900
  1:  c=deck     UelKmDef900
  2:  c=purpose  Compute material stiffness matrix for user-written E900
  3:  c=author   --------------------------------------------------------
  5:  c=author   --------------------  Ph: (858)259-2773 or (858)481-9907
  6:  c=version  December 2001
  7:
  8:  #if _usage_
  9:  *
10:  *     call UelKmDef900 (type,   elt,   nlin, iplast, nnodes,
11:  *                        active, nvars, xe,   de,     km)
12:  *
13:  * Input Arguments
14:  *
15:  * type    =  element type (=900)
16:  * elt     =  element number of type TYPE in the model
17:  * nlin    =  0 - do not use nonlinear strain/displ. relations
18:  *        =  1 - use nonlinear relations for applicable elements
19:  * iplast  =  0 - do not use plasticity theory
20:  *        =  1 - use plasticity theory
21:  * nnodes  =  number of element nodes
22:  * active  =  number of active element nodes
23:  * nvars   =  number of element variables
24:  * xe      =  element local coordinates
25:  * de      =  element local deformation vector
26:  *
27:  * Output Arguments
28:  *
29:  * km      =  material stiffness matrix
30:  *
31:  #endif
32:
33:  ************************************************************************
34:        Subroutine UelKmDef900 (type,   elt,   nlin, iplast, nnodes,
35:       &                        active, nvars, xe,   de,     km)
36:  ************************************************************************
37:
38:  #     include   "keydefs.h"
39:```

User beam material stiffness matrix definition routine—UelKmDef900

The purpose of a User element material stiffness matrix definition routine (such as UelKmDef900) is to compute and return the material stiffness matrix for a specific UEL. Each User element material stiffness matrix definition routine called by the UelKmDef dispatcher is provided for this purpose. The material stiffness matrix definition routine for this User element is presented below (with line numbers shown on the left for reference purposes):
40:        _implicit_none_
41:
42:  #     include   "stndcm.h"
43:
44:        Integer    type
45:        Integer    elt
46:        Integer    nlin
47:        Integer    iplast
48:        Integer    nnodes
49:        Integer    active
50:        Integer    nvars
51:        _float_    xe(3,nnodes)
52:        _float_    de(nvars)
53:        _float_    km(*)
54:
55:  *     TO BE IMPLEMENTED BY THE USER
56:  *     =============================
57:
58:        Integer    size
59:
60:        _float_    A
61:        _float_    E
62:        _float_    G
63:        _float_    Iy
64:        _float_    Iz
65:        _float_    J
66:        _float_    L
67:        _float_    Mat
68:        _float_    SDy
69:        _float_    SDz
70:
71:        _float_    K0101
72:        _float_    K0202
73:        _float_    K0206
74:        _float_    K0303
75:        _float_    K0305
76:        _float_    K0404
77:        _float_    K0505
78:        _float_    K0511
79:        _float_    K0606
80:        _float_    K0612
81:
82:  *     Retrieve Element And Material Properties
83:  *     ----------------------------------------
84:        call UelGetFloVar (type,elt,'Area',      A,   size)
85:        call UelGetFloVar (type,elt,'Iy',        Iy,  size)
86:        call UelGetFloVar (type,elt,'Iz',        Iz,  size)
87:        call UelGetFloVar (type,elt,'J',         J,   size)
88:        call UelGetFloVar (type,elt,'Length',    L,   size)
89:        call UelGetFloVar (type,elt,'Material',  Mat, size)
90:        call UelGetFloVar (type,elt,'ShearDefY', SDy, size)
91:        call UelGetFloVar (type,elt,'ShearDefZ', SDz, size)
call UelGetFloProp (nint(Mat),'E', E, size)
call UelGetFloProp (nint(Mat),'G', G, size)

* Stiffness Coefficients

K0101 = E*A/L

K0202 = 12*E*Iz/((1+SDy)*L**3)
K0206 = 6*E*Iz/((1+SDy)*L**2)

K0303 = 12*E*Iy/((1+SDz)*L**3)
K0305 = 6*E*Iy/((1+SDz)*L**2)

K0404 = G*J/L

K0505 = (4+SDz)*E*Iy/((1+SDz)*L)
K0511 = (2-SDz)*E*Iy/((1+SDz)*L)

K0606 = (4+SDy)*E*Iz/((1+SDy)*L)
K0612 = (2-SDy)*E*Iz/((1+SDy)*L)

* Material Stiffness Matrix

km( 1) =  K0101
km(22) = -K0101
km( 3) =  K0202
km(17) =  K0206
km(30) = -K0202
km(68) =  K0206
km( 6) =  K0303
km(13) = -K0305
km(39) = -K0303
km(58) = -K0305
km(10) =  K0404
km(49) = -K0404
km(15) =  K0505
km(41) =  K0305
km(60) =  K0511
km(21) =  K0606
km(34) = -K0206
km(72) =  K0612
km(28) =  K0101
km(36) =  K0202
km(74) =  K0206
Lines 1–57 are provided as part of the standard **UelKmDef900** template routine. The idea of any template is to provide developers all the data that they might need to calculate the requested output or internal intermediate data.

Lines 58–80 define all the variables that are needed for the local operations. This routine is going to compute an element’s material stiffness matrix.

Lines 84–91 retrieve some floating-point User element variables for this specific type 900 element. Notice that the names in the **UelGetFloVar** calls were created by the user in the *userElement* directive in the model description file. The material identification parameter (Material) is used as a User property set identifier on lines 93–94 to retrieve the material properties for this particular element.

Computations on lines 98–112 are performed to calculate the beam element’s stiffness coefficients. These stiffness values are then used on lines 116–152 to set the element’s material stiffness matrix components. Notice that only the nonzero components need to be set since the matrix work space has been initialized to zero by **STAGS**. While these computations are very simple, they do demonstrate the general approach for any User element material stiffness matrix definition routine.
User beam strain vector definition routine—UelStrainDef900

The purpose of a User element strain vector definition routine (such as UelStrainDef900) is to compute and return the strain vector for a specific User element at a number of sampling points. Each User element strain vector definition routine called by the UelStrainDef dispatcher is provided for this purpose. The strain vector definition routine for this User element is presented below (with line numbers shown on the left for reference purposes):

```fortran
subroutine UelStrainDef900
  c=deck     UelStrainDef900
  c=purpose  Compute strain vector for user-written element E900
  c=author   Lyle W. Swenson, Jr.  Knowledge Management Systems, Inc.
  c=version  December 2001
  #if _usage_
  *     call UelStrainDef900 (type,   elt,    nlin,  iplast,
  *                       nnodes, active, nvars, nips,
  *                       neps,   ix,     xe,    de,
  *                       strain)
  #endif

1:  c=deck     UelStrainDef900
2:  c=purpose  Compute strain vector for user-written element E900
4:  c=version  December 2001
5:  #if _usage_
6:  *     call UelStrainDef900 (type,   elt,    nlin,  iplast,
7:  *                       nnodes, active, nvars, nips,
8:  *                       neps,   ix,     xe,    de,
9:  *                       strain)
10: *     Input Arguments
11: *     type    =  element type (=900)
12: *     elt     =  element number of type TYPE in the model
13: *     nlin    =  =0 - do not use nonlinear strain/displ. relations
14: *                       =1 - use nonlinear relations for applicable elements
15: *     iplast  =  =0 - do not use plasticity theory
16: *                       =1 - use plasticity theory
17: *     nnodes  =  number of element nodes
18: *     active  =  number of active element nodes
19: *     nvars   =  number of element variables
20: *     nips    =  number of strain sampling points
21: *     neps    =  number of strain components
22: *     ix      =  node permutation array
23: *     xe      =  nodal coordinates in element system
24: *     de      =  element local deformation vector
25: *     strain  =  element strain vector at each sampling point
26: #endif
```

Subroutine UelStrainDef900 (type,   elt,    nlin,  iplast,
User–Defined Elements Uniform Beam Example

& nnodes, active, nvars, nips,
& neps, ix, xe, de,
& strain)
************************************************************************
#
#     include   "keydefs.h"
#
implicit_none_
#
#     include   "stndcm.h"

Integer    type
Integer    elt
Integer    nlin
Integer    iplast
Integer    nnodes
Integer    active
Integer    nvars
Integer    nips
Integer    neps
Integer    ix(active)
_float_    xe(3,nnodes)
_float_    de(nvars)
_float_    strain(neps,nips)

* TO BE IMPLEMENTED BY THE USER
* =============================

_float_    E
_float_    G
_float_    Mat
_Float_    Pr

Integer    i
Integer    n
_Float_    s(6)
Integer    size

* Retrieve Element And Material Properties
* ----------------------------------------
 call UelGetFloVar (type,elt,'Material', Mat, size)
 call UelGetFloProp (nint(Mat),'E', E, size)
 call UelGetFloProp (nint(Mat),'G', G, size)
 call UelGetFloProp (nint(Mat),'PoissonRatio', Pr, size)

* Compute Beam Element Stresses At Each Sampling Point
* ----------------------------------------------------
 call UelStressDef900 (type, elt, nlin, iplast,
 & nnodes, active, nvars, nips,
 & neps, ix, xe, de,
& strain)

* Compute Beam Element Strains At Each Sampling Point
* ---------------------------------------------------
* +------------+
* | Strains:   |
* |   1. exx   |
* |   2. eyy   |
* |   3. ezz   |
* |   4. exy   |
* |   5. exz   |
* |   6. eyz   |
* +------------+

n = nips/2

do 10 i=1,n

* First Node...
* ------------
call scopud (6,strain(1,i), s)

strain(1,i) = ( s(1) - Pr * (s(2) + s(3)) ) / E
strain(2,i) = ( s(2) - Pr * (s(3) + s(1)) ) / E
strain(3,i) = ( s(3) - Pr * (s(1) + s(2)) ) / E
strain(4,i) = s(4) / G
strain(5,i) = s(5) / G
strain(6,i) = s(6) / G

* Second Node...
* -------------
call scopud (6,strain(1,i+n), s)

strain(1,i+n) = ( s(1) - Pr * (s(2) + s(3)) ) / E
strain(2,i+n) = ( s(2) - Pr * (s(3) + s(1)) ) / E
strain(3,i+n) = ( s(3) - Pr * (s(1) + s(2)) ) / E
strain(4,i+n) = s(4) / G
strain(5,i+n) = s(5) / G
strain(6,i+n) = s(6) / G

10 continue
end

Lines 1–67 are provided as part of the standard UelStrainDef900 template routine. The idea of any template is to provide developers all the data they might need to calculate the requested output or internal intermediate data.

Lines 68–76 define all the variables that are needed for the local operations. This routine is going to compute an element’s strain vector at a number of sampling points.
Line 80 retrieves the material identifier for this specific type 900 element. Notice that the names in the UelGetFloVar calls were created by the user in the *userElement directive in the model description file. The material identification parameter (Material) is used as a User property set identifier in lines 82–84 to retrieve the material properties for this particular element.

Because of its formulation, this particular User element computes element stresses from element internal forces (using UelStressDef900) and element strains from the element stresses. The element’s stresses are computed via the call to UelStressDef900 on lines 89–92. Notice that this stress routine invocation is possible because all of its required arguments are also arguments passed to the strain calculation routine. The stress values are then used on lines 106–132 to set the element’s strain components. While these computations are very simple, they do demonstrate the general approach for any User element strain vector definition routine.
User beam stress vector definition routine—UelStressDef900

The purpose of a User element stress vector definition routine (such as UelStressDef900) is to compute and return the stress vector for a specific UEL at a number of sampling points. Each User element stress vector definition routine called by the UelStressDef dispatcher is provided for this purpose. The stress vector definition routine for this User element is presented below (with line numbers shown on the left for reference purposes):

```fortran
subroutine UelStressDef900
  c=deck     UelStressDef900
  c=purpose  Compute stress vector for user-written element E900
  c=author   --------------------------------------------------------
  c=author   Lyle W. Swenson, Jr.  Knowledge Management Systems, Inc.
  c=author   --------------------  Ph: (858)259-2773 or (858)481-9907
  c=version  December 2001
  
  #if _usage_

  *     call UelStressDef900 (type,   elt,    nlin,  iplast,
  *                           nnodes, active, nvars, nips,
  *                           nsig,   ix,     xe,    de,
  *                           stress)
  *
  Input Arguments
  type    =  element type (=900)
  elt     =  element number of type TYPE in the model
  nlin    =  =0 - do not use nonlinear strain/displ. relations
            =1 - use nonlinear relations for applicable elements
  iplast  =  =0 - do not use plasticity theory
            =1 - use plasticity theory
  nnodes  =  number of element nodes
  active  =  number of active element nodes
  nvars   =  number of element variables
  nips    =  number of stress sampling points
  nsig    =  number of stress components
  ix      =  node permutation array
  xe      =  nodal coordinates in element system
  de      =  element local deformation vector
  
  Output Arguments
  stress  =  element stress vector at each sampling point

  Subroutine UelStressDef900 (type,   elt,    nlin,  iplast,
```

User–Defined Elements

Uniform Beam Example

40:        &
41:        &
42:        &
43:        &
44:        &
45:        &
46:        &
47:        &
48:        &
49:        &
50:        &
51:        &
52:        &
53:        &
54:        &
55:        &
56:        &
57:        &
58:        &
59:        &
60:        &
61:        &
62:        &
63:        &
64:        &
65:        &
66:        &
67:        &
68:        &
69:        &
70:        &
71:        &
72:        &
73:        &
74:        &
75:        &
76:        &
77:        &
78:        &
79:        &
80:        &
81:        &
82:        &
83:        &
84:        &
85:        &
86:        &
87:        &
88:        &
89:        &
90:        &
91:        &

TO BE IMPLEMENTED BY THE USER

Retrieve Element And Material Properties

call UelGetFloVar (type, elt, 'Area', A, size)
call UelGetFloVar (type, elt, 'Iy', Iy, size)
92:        call UelGetFloVar (type, elt, 'Iz', Iz, size)
93:        call UelGetFloVar (type, elt, 'J', J, size)
94:        call UelGetFloVar (type, elt, 'ShearFactorT', SFt, size)
95:        call UelGetFloVar (type, elt, 'ShearFactorY', SFy, size)
96:        call UelGetFloVar (type, elt, 'ShearFactorZ', SFz, size)
97:
98:  *     Retrieve Stress Sampling Points
99:  *     -------------------------------
100:        call UelGetFloProp (type, 'y1', y(1), size)
101:        call UelGetFloProp (type, 'y2', y(2), size)
102:        call UelGetFloProp (type, 'y3', y(3), size)
103:        call UelGetFloProp (type, 'y4', y(4), size)
104:        call UelGetFloProp (type, 'y5', y(5), size)
105:
106:        call UelGetFloProp (type, 'z1', z(1), size)
107:        call UelGetFloProp (type, 'z2', z(2), size)
108:        call UelGetFloProp (type, 'z3', z(3), size)
109:        call UelGetFloProp (type, 'z4', z(4), size)
110:        call UelGetFloProp (type, 'z5', z(5), size)
111:
112:  *     Compute Beam Element Internal Forces -- FI
113:  *     ------------------------------------------
114:        istep = 0
115:        pa = 0.0
116:        pb = 0.0
117:
118:        call UelFiDef900 (type, elt, nlin, iplast, istep,
119:          & pa, pb, active, nvars, ix,
120:          & de, fi, iplmax, energy)
121:
122:  *     Compute Beam Element Stresses At Each Sampling Point
123:  *     ----------------------------------------------------
124:  *     +------------+
125:  *     | Stresses:  |
126:  *     |  1. sxx    |
127:  *     |  2. syy    |
128:  *     |  3. szz    |
129:  *     |  4. sxy    |
130:  *     |  5. sxz    |
131:  *     |  6. syz    |
132:  *     +------------+
133:
134:        n = nips/2
135:
136:        do 10 i=1,n
137:  138:  *     First Node...
139:  *     -----------------
140:        stress(1,i) = -fi(1)/A - fi(5)*z(i)/Iy + fi(6)*y(i)/Iz
141:        stress(2,i) = 0.0
142:        stress(3,i) = 0.0
144:      if (i.ne.n) then
145:         stress(4,i) = fi(4)*z(i)*SFt/J
146:         stress(5,i) = -fi(4)*y(i)*SFt/J
147:         stress(6,i) =  0.0
148:      else
149:         stress(4,i) = -fi(2)*SFy/A
150:         stress(5,i) = -fi(3)*SFz/A
151:         stress(6,i) =  0.0
152:      endif

153:
154:      *        Second Node...
155:      *
156:      *        --------------
157:      stress(1,i+n) = fi(7)/A + fi(11)*z(i)/Iy - fi(12)*y(i)/Iz
158:      stress(2,i+n) = 0.0
159:      stress(3,i+n) = 0.0
160:      if (i.ne.n) then
161:         stress(4,i+n) = -fi(10)*z(i)*SFt/J
162:         stress(5,i+n) = fi(10)*y(i)*SFt/J
163:         stress(6,i+n) =  0.0
164:      else
165:         stress(4,i+n) = fi(8)*SFy/A
166:         stress(5,i+n) = fi(9)*SFz/A
167:         stress(6,i+n) = 0.0
168:      endif

169:
170:    10    continue
171:    
172:    end

Lines 1–67 are provided as part of the standard **UelStressDef900** template routine. The idea of any template is to provide developers all the data they might need to calculate the requested output or internal intermediate data.

Lines 68–86 define all the variables that are needed for the local operations. This routine computes an element’s stress vector at a number of sampling points.

Lines 90–96 retrieve some floating-point User element variables for a specific type 900 element. Notice that the names in the **UelGetFloVar** calls were created by the user in the **userElement** directive in the model description file.

Lines 100–110 retrieve the sampling points provided in the User property set tagged with the same identifier as the User element identifier. Recall that the sampling points could have been placed in any property set the user might have created—not necessarily the property set (i.e., the one with id=900) with the parameters required for the User element type.
This particular User element computes element stresses from element internal force components given the element’s deformations (using \texttt{UelFiDef900}). The element’s internal forces are computed \textit{via} the call to \texttt{UelFiDef900} on lines 118–120. Notice that this internal force invocation is possible because most of its required arguments are also arguments passed to the stress calculation routine. The internal force values are then used on lines 140–170 to compute the element’s stress components. While these computations are very simple, they do demonstrate the general approach for any User element stress vector definition routine.
User beam strain printing definition routine—UelPrintStrainDef900

The purpose of a User element strain printing definition routine (such as UelPrintStrainDef900) is to print the strain vector for a specific User element at a number of sampling points. Each User element strain printing routine called by the UelPrintStrainDef dispatcher is provided for this purpose. The strain printing definition routine for this User element is presented below (with line numbers shown on the left for reference purposes):

```fortran
subroutine UelPrintStrainDef900
1:  c=deck     UelPrintStrainDef900
2:  c=purpose  Print strain vector for a user-written element E900
3:  c=author   --------------------------------------------------------
5:  c=author   --------------------  Ph: (858)259-2773 or (858)481-9907
6:  c=version  December 2001
7:  
8:  #if _usage_
9:  * 
10:  *     call UelPrintStrainDef900 (type,   elt,      nnodes, active,
11:  *                                nips,   neps,     ix,     xe,
12:  *                                strain, maxLines, lines)
13:  * 
14:  *     Input Arguments
15:  *     ----------------
16:  *     type      =  element type (=900)
17:  *     elt       =  element number of type TYPE in the model
18:  *     nnodes    =  number of element nodes
19:  *     active    =  number of active element nodes
20:  *     nips      =  number of strain sampling points
21:  *     neps      =  number of strain components
22:  *     ix        =  node permutation array
23:  *     xe        =  nodal coordinates in element system
24:  *     strain    =  element strain vector at each sampling point
25:  *     maxLines  =  number of lines per “page”
26:  *     lines     =  number of lines printed prior to the call
27:  * 
28:  *     Output Arguments
29:  *     ----------------
30:  *     lines     =  number of lines printed after the call
31:  * 
32:  #endif
33:  
34:  ************************************************************************
35:        Subroutine UelPrintStrainDef900 (type,   elt,      nnodes, active,
36:       &                                nips,   neps,     ix,     xe,
37:       &                                strain, maxLines, lines)
38:  ************************************************************************
39: 
```

40:     include  "keydefs.h"
41:  _implicit_none_
42:  include  "stndcm.h"
43:    Integer    type
44:    Integer    elt
45:    Integer    nnodes
46:    Integer    active
47:    Integer    nips
48:    Integer    neps
49:    Integer    ix(active)
50:    _float_    xe(3,nnodes)
51:    _float_    strain(neps,nips)
52:    Integer    maxLines
53:    Integer    lines
54:  *     TO BE IMPLEMENTED BY THE USER
55:  *     =============================
56:    Integer    i
57:    Integer    j
58:  *     CHECK FOR NEW HEADER
59:  *     --------------------
60:    if (lines+nips .gt. maxLines) then
61:       write(not,100) type
62:       lines = 6
63:    endif
64:  *     PRINT STRAINS AT EACH SAMPLING POINT
65:  *     ------------------------------------
66:    do 10 j=1,nips
67:       write(not,200) elt, j, (strain(i,j),i=1,neps)
68:  10    continue
69:    write(not,300)
70:    lines = lines + nips + 1
71:    format(/'--------------------------------'
72:       &' Strain for User Element Type',i4
73:       &' Element Point',7x,'Exx','Eyy','Ezz',11x,'Exy',11x,'Exz',11x,'Eyz'
74:       &' -------  -----',6('  ------------'))
75:    format(i8,i7,1p6e14.5)
76:    200 format(' ')
77:    300 format(' ')
78:    end
Lines 1–60 are provided as the core part of the standard UelPrintStrainDef900 template routine. The idea of any template is to provide developers all the data they might need to calculate the requested output or internal intermediate data.

The templates that are supplied with STAGS for printing strain (i.e., strain printing definition routines UelPrintStrainDef9xx) also include a simple implementation to provide users a head-start on specialization. The sample implementation is described in the remainder of this section.

Lines 61–62 define all the variables needed for the local operations.

Lines 66–69 check to determine if a new header needs to be printed before the strain components are printed and prints the header if it is required.

Lines 73–75 print the element’s strain components at each of the sampling points.

Finally, lines 77–78 print a spacer line and increments the output line count argument by the number of lines just printed.

The sample strain printing implementations assume that there are six (6) strain components with a certain ordering (i.e., Exx, Eyy, Ezz, Exy, Exz and Eyz). If your User element strain calculations meet these conditions, then the UelPrintStrainDef9xx templates can be used as they are. Otherwise, only minor modifications will probably be required. For this example, no modifications to the UelPrintStrainDef900 template were required.
Beam stress printing definition routine—UelPrintStressDef900

The purpose of a User element stress printing definition routine (like UelPrintStressDef900) is to print the stress vector for a specific UEL at a number of sampling points. Each User element stress printing routine called by the UelPrintStressDef dispatcher is provided for this purpose. The stress printing definition routine for this User element is presented below (with line numbers shown on the left for reference purposes):

```
subroutine UelPrintStressDef900
  c=deck  UelPrintStressDef900
  c=purpose Print stress vector for a user-written element E900
  c=author ---------------------------------------------
  c=author Lyle W. Swenson, Jr. Knowledge Management Systems, Inc.
  c=author --------------------- Ph: (858)259-2773 or (858)481-9907
  c=version December 2001
  
  #if _usage_
  *
  *    call UelPrintStressDef900 (type, elt, nnodes, active,
  *   nips, nsig, ix, xe,
  *   stress, maxLines, lines)
  *
  *    Input Arguments
  *
  *    type      =  element type (=900)
  *    elt       =  element number of type TYPE in the model
  *    nnodes    =  number of element nodes
  *    active    =  number of active element nodes
  *    nips      =  number of stress sampling points
  *    nsig      =  number of stress components
  *    ix        =  node permutation array
  *    xe        =  nodal coordinates in element system
  *    stress    =  element stress vector at each sampling point
  *    maxLines  =  number of lines per “page”
  *    lines     =  number of lines printed prior to the call
  *
  *    Output Arguments
  *
  *    lines     =  number of lines printed after the call
  *
  #endif
```

---

**subroutine UelPrintStressDef900**

```
1: c=deck     UelPrintStressDef900
2: c=purpose  Print stress vector for a user-written element E900
3: c=author   ---------------------------------------------
5: c=author   --------------------- Ph: (858)259-2773 or (858)481-9907
6: c=version  December 2001
7:
8: #if _usage_
9: *
10: *    call UelPrintStressDef900 (type, elt, nnodes, active,
11: *    nips, nsig, ix, xe,
12: *    stress, maxLines, lines)
13: *
14: *    Input Arguments
15: *    ---------------------------------------------
16: *    type      =  element type (=900)
17: *    elt       =  element number of type TYPE in the model
18: *    nnodes    =  number of element nodes
19: *    active    =  number of active element nodes
20: *    nips      =  number of stress sampling points
21: *    nsig      =  number of stress components
22: *    ix        =  node permutation array
23: *    xe        =  nodal coordinates in element system
24: *    stress    =  element stress vector at each sampling point
25: *    maxLines  =  number of lines per “page”
26: *    lines     =  number of lines printed prior to the call
27: *
28: *    Output Arguments
29: *    ---------------------------------------------
30: *    lines     =  number of lines printed after the call
31: *
32: #endif
```

---

**Subroutine UelPrintStressDef900**

```
Subroutine UelPrintStressDef900 (type, elt, nnodes, active,
  & nips, nsig, ix, xe,
  & stress, maxLines, lines)
```
# include "keydefs.h"

implicit none

# include "stndcm.h"

Integer type
Integer elt
Integer nnodes
Integer active
Integer nips
Integer nsig
Integer ix(active)
_float_ xe(3,nnodes)
_float_ stress(nsig,nips)
Integer maxLines
Integer lines

* TO BE IMPLEMENTED BY THE USER
* =============================

Integer i
Integer j

* CHECK FOR NEW HEADER
* ---------------------
if (lines+nips .gt. maxLines) then
   write(not,100) type
   lines = 6
endif

* PRINT STRESS AT EACH SAMPLING POINT
* -----------------------------------
do 10 j=1,nips
   write(not,200) elt, j, (stress(i,j),i=1,nsig)
10    continue

write(not,300)
   lines = lines + nips + 1

100   format(/' --------------------------------'
   &       /' Stress for User Element Type',i4
   &       /' --------------------------------'
   &      //' Element  Point',7x,'Sxx',11x,'Syy',11x,'Szz',11x,
   &        'Sxy',11x,'Sxz',11x,'Syz'
   &       /' -------  -----',6('  ------------'))

200   format(i8,i7,1p6e14.5)

300   format(' ')
end
Lines 1–60 are provided as the core part of the standard `UelPrintStressDef900` template routine. The idea of any template is to provide developers all the data they might need to calculate the requested output or internal intermediate data.

The templates that are provided with STAGS for printing stress (i.e., stress printing definition routines `UelPrintStressDef9xx`) also include a simple implementation to provide users a head-start on specialization. The sample implementation is described in the remainder of this section.

Lines 61–62 define all the variables needed for the local operations.

Lines 66–69 check to determine if a new header needs to be printed before the stress components are printed and prints the header if required.

Lines 73–75 print the element’s stress components at each of the sampling points.

Finally, lines 77–78 print a spacer line and increments the output line count argument by the number of lines just printed.

The sample stress printing implementations assume that there are six (6) stress components with a certain ordering (i.e., $S_{xx}$, $S_{yy}$, $S_{zz}$, $S_{xy}$, $S_{xz}$ and $S_{yz}$). If your User element stress calculations meet these conditions, then the `UelPrintStressDef9xx` templates can be used as they are. Otherwise, only minor modification will probably be required. For this example, no modifications to the `UelPrintStressDef900` template were required.

Analysis and Results

A nonlinear static analysis was performed for the beam model and User element described above. The STAGS solution description file (`ubeam1.bin`) is given below (with line numbers on the left shown for reference purposes):

```
ubeam1.bin

1:  User Beam Nonlinear Analysis
2:  3 1 1 0 0 0 0 1 $ b1 -- indic,ipost,ilist,icor,...,isolvr
3:  1 0 0 $ b2 -- icpact,iter,iprim,...
4:  500.0 500.0 500000.0 $ c1 -- stld,step,facm,...
5:  0 0 5 -20 0 1.0e-6 $ d1 -- istart,nsec,ncut,newt,nstrat,delx,...
6:  0 $ et1 -- npath,...
```
The applied load magnitudes (500,000 lb) are highly unrealistic here (i.e., they are extremely large). These values were chosen to create large nonlinear displacements and rotations and to demonstrate the efficacy of STAGS’ corotation architecture. No special work is required on the part of users or developers to benefit from these powerful corotational features. Although linear element formulations will not suffice for all geometrically nonlinear problems, there are many situations were materially linear elements with automatic corotational corrections yield excellent results.

A load versus displacement plot is shown in the following Figure for the beam tip in the z–direction. For comparison purposes the plot also includes the result for the same problem using STAGS’ E210 beam elements. Although the E210 element uses a nonlinear strain-displacement formulation, for this problem the results are virtually indistinguishable.
14

The Element Library

14.1 Organization

This Chapter contains a very brief (summary) description of each of the “standard” and “special-purpose” elements that are implemented in the current version of the STAGS program. Element definition via User-written subroutines is discussed in Chapter 12 of this document, and the construction and utilization of User-defined elements is discussed (briefly) in Chapter 13. The STAGS Elements Manual contains fuller descriptions and discussions of many (but not all) of the standard and special-purpose elements that are summarized here. It also contains essential information about the construction and utilization of User-defined elements. As in the STAGS Elements Manual, the standard and special-purpose STAGS elements are organized here in six main groups:

- “spring” elements
- “beam” elements
- shell and mesh-transition shell elements
- sandwich and mesh-transition sandwich elements
- solid elements
- contact elements

User-defined elements are described in the preceding Chapter and in the STAGS Elements Manual, and will not be discussed here. The discussion of each element is accompanied by a figure that contains a graphical representation of that element. Some common symbols used in the figures are:

- \( n \) – node \( n \)
- \( n \) – side \( n \)
- \( n \) – integration point \( n \)
14.2 Algorithm for Determining the Element Frame

The origin of the element frame \((x', y', z')\) is located at node 1, with the nodes numbered in counterclockwise order, corner nodes first. Although this numbering is strictly adhered to, for element units the origin (node 1) is selected by STAGS to provide the best possible local frame for the element computations. For this reason, the user should avoid stress and strain output in element coordinates, since the local element coordinate system can vary from element to element. As was stated before, the default STAGS option for output is in the shell-wall, or \((\tilde{x}, \tilde{y})\) system, thereby avoiding any reference to the element system. By a suitable definition \((x, y)\) for the wall and \((\phi_1, \phi_2)\) for the material layers, the user can select whatever coordinate system he wants for the output of these quantities.

The actual algorithm for triangles and quadrilaterals is now described, with \((x_{g1}, y_{g1}, z_{g1})\) representing the updated or current global coordinates of the nodal points when corotation is invoked (default). For triangles, \(x'\) points from node 1 to node 2. The following formulas determine the orientation of the axes:

\[
\begin{align*}
x' &= \frac{x_{g2} - x_{g1}}{|x_{g2} - x_{g1}|} \\
r &= x_{g3} - x_{g1} \\
z' &= \frac{x' \times r}{|x' \times r|} \\
y' &= z' \times x'
\end{align*}
\]

(14.1)

where the numerical subscripts refer to the three corner nodes of the triangle. For quadrilaterals, we first determine the normal as the cross product of the diagonals:

\[
z' = \frac{(x_{g3} - x_{g1}) \times (x_{g4} - x_{g2})}{|(x_{g3} - x_{g1}) \times (x_{g4} - x_{g2})|}
\]

(14.2)

The other two axes are determined by

\[
\begin{align*}
r &= x_{g4} - x_{g1} \\
x' &= \frac{r \times z'}{|r \times z'|} \\
y' &= z' \times x'
\end{align*}
\]

(14.3)

For beams, we first determine the initial element system using an auxiliary node. This auxiliary node is along the shell normal with its origin at node 1 (in the case of shell units),
or it is given by the user (element units). In all cases, \( x' \) is determined by the vector joining the endpoints of the beam:

\[
x' = \frac{x_{g_2} - x_{g_1}}{\left| x_{g_2} - x_{g_1} \right|}
\]  

(14.4)

For shell units, the initial system follows

\[
\begin{align*}
  r &= x_{g_3} - x_{g_1} \\
  y' &= \frac{r \times x'}{|r \times x'|} \\
  z' &= x' \times y'
\end{align*}
\]  

(14.5)

whereas for the element units

\[
\begin{align*}
  r &= x_{g_3} - x_{g_1} \\
  z' &= \frac{x' \times r}{|x' \times r|} \\
  y' &= z' \times x'
\end{align*}
\]  

(14.6)

or the same as for triangles (14.1). In order to obtain the element system for a current configuration using the corotational option, it is necessary to account for both the translation and rotation of the two endpoint nodes. Initially, the nodal orientation for both nodes is taken to be the same as in the initial element system. In subsequent steps in the analysis, both the coordinates and the nodal orientations change. STAGS keeps track of the nodal orientations by storing the rotation \( T \) necessary to take the node from its initial position to the current position. The current orientation of node \( a \) is

\[
R_a = T_a E_0
\]  

(14.7)

where \( E_0 \) is the initial element system determined from (14.5) or (14.6). After the initial step, the beam system is determined as follows:
where the expression “int(\( R_1, R_2 \))” is shorthand for determining the triad that is the midpoint interpolation of triads \( R_1 \) and \( R_2 \), and where \( \hat{r}_i \) is the \( i \)th column of the matrix \( R_{1/2} \). The last equation of (14.8) is the result of rotating the interpolated \( \hat{r}_1 \) axis in \( R_{1/2} \) until it coincides with the line joining the endpoints of the beam element; there is a unique formula for rotating \( R_{1/2} \) the minimum possible amount until the two \( x \) axes coincide.
14.3 “Spring” Elements

The current version of the STAGS program has the following four $E_{100}$-series “spring” elements:

- the $E_{110}$ Mount element
- the $E_{120}$ Rigid Link element
- the $E_{121}$ Soft Link element
- the $E_{130}$ Generalized Fastener element

For geometric reasons, all of these elements are called “spring” elements. The $E_{120}$ (rigid link) element is considered to be a standard element; but the $E_{110}$ (mount), $E_{121}$ (soft link) and $E_{130}$ (generalized fastener) elements are considered to be special-purpose—for reasons that should be clear from the descriptions in the next four subsections.
E110 Mount element

The E110 mount element, shown in Figure 14.1, uses a special nonlinear spring that is capable of modeling a User-defined displacement-velocity-force profile. The action of the force resulting from the extension and/or motion of the nonlinear spring's end-points can be applied with an offset from the element's nodes. This is achieved using rigid links connecting the element's nodes (denoted N1 and N2) and the nonlinear spring's end-points (see Figure 14.1). Rigid links provide a method for defining rotational stiffness in addition to axial spring stiffness. Geometry for the node N1 and node N2 rigid links is expressed in the node's local element coordinate system (X1,Y1,Z1) and (X2,Y2,Z2), respectively, as shown in Figure 14.1. The coordinate X1 is in the direction of the line connecting N1 and N2. The Y1 axis is normal to X1 and in the plane defined by N1, N2, and N3, completing a right-handed system. (Clearly, node N3 must not lie on the same line as N1 and N2.) N3 can be either a structural node or a dummy node, defined only for reference to the mount. Any freedoms defined on dummy nodes are ignored. For rigid link 1, the origin of the (X1,Y1,Z1) system is situated at N1, and the distance RL1x, RL1y, and RL1z are expressed in this system. For rigid link 2, the (X2,Y2,Z2) system, with origin at N2, is used to express the distances RL2x, RL2y, and RL2z. The component values can be positive, negative, or zero. When all components for a rigid link are zero, the rigid link does not exist. See Figure 14.1.

Figure 14.1  E110 Mount element
E120 Rigid link element

The E120 rigid link element, shown in Figure 14.2, constrains the distance between two nodes (N1 and N2) to be an invariant during an analysis. The displacements of node N2 are dependent on the displacements and rotations of node N1 through a rigid-link constraint equation, which is enforced via Lagrange multipliers (see Figure 14.2), and the more complete discussion of this element in the STAGS Elements Manual. Note that the rotations for N2 are not constrained by the motion of N1; if N2 rotations are to be constrained, then the constraints must be explicitly specified, or the reference node N3 must have a nonzero value. In the latter case, STAGS will generate partial compatibility constraints (see records G-2) constraining the rotations of N2 to be the same as those of N1.

Figure 14.2 E120 Rigid link element

E121 Soft link element

The three-node E121 soft link element, shown in Figure 14.3, has been designed for use at junctions between surfaces idealized by shell elements and assemblages of solid elements. The E121 element forces two solid-element nodes (N1 and Nk) at a junction to remain “straight,” or aligned along the shell normal \( \hat{z} \) of the shell node N0 to which they are joined. The shell normal, in turn, rotates rigidly with the shell node in response to equilibrium. The position of the shell node can be described as a linear weighted average of the solid nodes through the thickness. The solid element nodes are free to move along the shell normal—allowing for stress relief in the thickness direction, consistent with a plane-stress
approximation. For a full discussion of this element, please see the STAGS Elements Manual.

E130 Generalized fastener element

The E130 fastener element is shown in Figure 14.4:

Figure 14.3  E121 Soft link element

Figure 14.4  Fastener element local coordinates, displacements, and rotations
The **E130** element is a generalization of the **E110** Mount with the provision of *six* hyperelastic and/or elastic-plastic spring functions (see Figure 7.1 “Degree-of-Freedom Directions at an Auxiliary Node” on page 7-6), with a potentially different function for each of the three local translations and rotations. The local directions are illustrated in Figure 14.4 on page 14-8, where we find that the local x axis lies along the line between the first two nodes, labeled N1 and N2 in the figure. The local z axis is normal to the plane determined by N1, N2, and the reference node N3. The y axis completes a right-hand orthogonal system. Before deformation, the direction vector along the line between N1 and N2 is split evenly into two *rigid links* of length \( \frac{L}{2} \). Local deformational rotation and translation of each node brings these links into new positions illustrated in the top half of the picture. The translational deformations are labeled near the center of the triangle, where the open circles denote the projection of the ends of the links onto the plane. If \( u_i \) and \( \theta_i \) are the local displacement and rotation vectors belonging to N1 or N2 for \( i = 1 \) or 2 respectively, then the displacement of the tip of the link is

\[
\mathbf{d}_i = \frac{\theta_i \times \mathbf{x}_{12}^i}{2} + \mathbf{u}_i
\]  

(14.9)

Here, \( \mathbf{x}_{12}^i \) is the vector that points *from* the node \( i \) to the other node, which also means that \( \mathbf{x}_{12}^2 = -\mathbf{x}_{12}^1 \). The bottom half of the figure shows the deformational angles for node N1. \( \dot{\theta}_x \) is the twist around the link in the x direction, \( \dot{\theta}_y \) is the counter-clockwise out-of-plane angle, and \( \dot{\theta}_z \) is the in-plane angle. The relative translations and rotations are

\[
\hat{\mathbf{d}} = \mathbf{d}_2 - \mathbf{d}_1 \\
\hat{\theta} = \theta_2 - \theta_1
\]  

(14.10)

The nodal displacements by this time have been passed through the same corotational software as is used for the **E210** beam. This insures that any rigid motion of the fastener has been eliminated, and consequently, that the deformational displacements are small for all practical cases.

The potential energy is a function only of the local displacements \( \hat{\mathbf{d}} \) and \( \hat{\theta} \):

\[
U(\hat{\mathbf{d}}, \hat{\theta}) = \sum_{i=1}^{3} [g_i(\hat{d}^i) + h_i(\hat{\theta}^i)]
\]  

(14.11)

where the index \( i \) refers to *components*. If the local freedoms are ordered as

\[
\hat{\mathbf{d}}^T = \begin{bmatrix} \hat{\mathbf{d}}^T \hat{\theta}^T \end{bmatrix},
\]  

(14.12)
and the nodal freedoms are ordered as

$$\bar{d}^T = \left\{ d_1^T \theta_1^T d_2^T \theta_2^T \right\}, \quad (14.13)$$

and if we define the moment arm as

$$L_{12} = \frac{x_{12}}{2}, \quad (14.14)$$

then equations (14.9) and (14.10) can be expressed as the matrix equation

$$\ddot{\bar{d}} = \ddot{\bar{d}} + \hat{P} \bar{d},
\hat{P} = \begin{bmatrix} -I & L_{12} & -L_{12} \\ 0 & -I & 0 & I \end{bmatrix}, \quad (14.15)$$

where we have used the tilde notation on the moment arms $L_{12}$ to denote the equivalent skew-symmetric matrix. If we take the first derivative of equation (14.11) and use the chain rule and equation (14.15), the nodal forces become

$$\hat{f} = \hat{P} \hat{d},
\hat{d} = \begin{bmatrix} g' \\ h' \end{bmatrix}, \quad (14.16)$$

where we have defined the six primitive spring forces from the first variation of equation (14.9) as a function of the local displacements:

$$g' = \frac{\partial g}{\partial \bar{d}}$$
$$h' = \frac{\partial h}{\partial \theta} \quad (14.17)$$

Since the potential energy itself is not needed to generate forces for the equilibrium equations, $g'$ and $h'$ can be defined directly by user input in the form of tables (or possibly from a User-written subroutine). Local forces for a particular point are linearly interpolated from the values in the table, and the local tangent stiffness is numerically computed from the derivatives of these forces and expressed as a 6x6 diagonal matrix $K$. From the chain rule and equations (14.15) and (14.16), we obtain the final nodal tangent stiffness:
Each relative displacement freedom can be matched up with a different nonlinear spring expressed as a Mount Table defined in the I-4 “material” records. At present, E130 fastener elements do not allow for velocity-dependent nonlinear spring response (if velocities are included in the table, the zero value is used). Otherwise, any force-deflection curve is acceptable. Type E130 elements must be referenced in an element unit.
14.4 “Beam” Elements

The current version of the STAGS program has the following two **E200-series** “beam” elements:

- the **E210** Beam element
- the **E250** Planar Boundary Condition element

For historical (and geometric) reasons, both of these elements are called “beam” elements. The **E210** (beam) element is a *standard* single- or multiple-component beam element; but the **E250** (Planar Boundary Condition) “element” is considered to be *special*—for reasons that should be clear from the descriptions in the next two subsections.
**E210 Beam element**

The **E210** beam element, shown in Figure 14.5, is “classic” in all senses of the word. Its cross-section geometry, specified via J-series Cross-Section records described in Chapter 5 of this document, is uniform along the length of the beam.

- nodes **N1**, **N2** and **N3** define the \((x', y')\) plane
- \(z'\) is normal to the \((x', y')\) plane
- \((x', y', z')\) – element coordinate system

**Figure 14.5**  **E210** Beam element
**E250 Planar boundary condition element**

The E250 Planar Boundary Condition “element”* in STAGS, shown in Figure 14.6, is STAGS’ response to the fact that there are situations that arise where the analyst wishes to constrain a section through or along a structure to remain in a specified plane but be free to move otherwise. This constraint can be visualized as an imaginary cut through the structure with

\[
(x', y', z') \quad \text{element coordinate system (see E210 Beam)}
\]

\[
(\hat{x}, \hat{y}, \hat{z}) \quad \text{updated nodal orientation coordinate system}
\]

\[
r_1 \quad \text{coordinates of node N1}
\]

\[
r_2 \quad \text{coordinates of node N2}
\]

\[
d_1 \quad \text{displacement at node N1}
\]

\[
d_2 \quad \text{displacement at node N2}
\]

A planar boundary condition is enforced by the following:

- \((\hat{x}, \hat{y}, \hat{z}) = (x', y', z')\) initially
- \(\hat{z}_1 \cdot ((r_2 + d_2) - (r_1 + d_1)) = 0\)
- \(\hat{z}_2 \cdot \hat{x}_1 = \hat{z}_2 \cdot \hat{y}_1 = 0\)

**Figure 14.6** E250 Planar boundary condition

the part that is on one side of the boundary modeled by FEM methods and the part that is on the other side of the boundary simulated by the planar (moving plane) boundary. Such a

moving plane is free to rotate and translate. A special case of this boundary is symmetry, where such a plane is constrained. An example of the moving plane is shown in the following figure:

![Diagram of moving plane]

**Description of the constraint**

The equation for the position of a point in a plane is

\[
\mathbf{n} \cdot \Delta \mathbf{p} = \mathbf{n}_1 \cdot \Delta \mathbf{p} = 0
\]

\[
\Delta \mathbf{p} = \mathbf{p}_2 - \mathbf{p}_1
\]

(14.19)

where \( \mathbf{n} \) is the normal to the plane, and \( \mathbf{p}_1 \) is already assumed to lie in the plane. It follows that \( \mathbf{p}_2 \) must also lie in the plane defined by \( \mathbf{n} \) if equation (14.19) is satisfied. One can see how a full boundary line can be constructed. First allow the first point \( \mathbf{p}_1 \) to define the position of the plane. If we apply equation (14.19) to this segment, \( \mathbf{p}_2 \) must also lie in the plane. By induction, we can continue the line as long as desired by enforcing additional points in succession. The only requirement is that these points must originally satisfy equation (14.19) in the *undeformed* configuration. This is achieved by using a third node (labeled \( \textbf{N3} \) in Figure 14.6), to determine the initial orientation of the plane containing \( \textbf{N1}, \textbf{N2} \) and \( \textbf{N3} \). Because of our ability to construct the entire boundary with individual coplanar segments that are linked together, we only need to consider one segment—shown in the figure above. For shell structures, we also have rotational freedoms that must leave any normal to the plane unchanged. This constraint is best achieved by the following construction:

![Diagram of normal constraint]
Let us assume that \( n_1 = n \) is normal to the plane. Then the following will allow free rotation about \( n_2 \) but will leave the normal at the new point \( p_2 \) unchanged:

\[
\begin{align*}
\hat{x} \cdot n_2 &= 0 \\
\hat{y} \cdot n_2 &= 0
\end{align*}
\] (14.20)

If we assume that the auxiliary vectors \( \hat{x} \) and \( \hat{y} \) rotate rigidly along with the normal \( n_1 \), the second normal \( n_2 \) will remain parallel to \( n_1 \) no matter how much that triad rotates. \( \hat{x} \) and \( \hat{y} \) are otherwise arbitrary and are easily constructed, for example, by using the vector joining the two nodes and completing a right hand system. Thus, equations (14.19) and (14.20) taken together are three nonlinear constraints that are valid no matter how large the subsequent motion of the system may be.

**Constraint kinematics**

The unit vectors \( \hat{x} \), \( \hat{y} \) and \( n_1 \) rotate rigidly in response to rotations at node \( p_1 \) from their initial positions \( \hat{x}_0 \), \( \hat{y}_0 \) and \( (n_1)_0 \), where the rotation of \( n_1 \) is possible because the plane in question may rotate. This rotation is expressed by a triad \( \Delta R \) formed from the product of rotations that are required to satisfy equilibrium. The kinematic equations are

\[
\begin{align*}
x &= \Delta R \hat{x}_0 \\
\hat{y} &= \Delta R \hat{y}_0 \\
n_1 &= \Delta R (n_1)_0 \\
\Delta R &\leftarrow \exp(\delta\omega)\Delta R
\end{align*}
\] (14.21)

where \( \delta\omega \) is the skew-symmetric (tilde symbol) matrix that is constructed from the rotation increments derived from the solution of the displacement equations, and where the last equation in this set expresses the update or correction of the triad from the previous iteration.

**Enforcement of the constraint**

Equations (14.19) and (14.20) can be enforced by adding the constraints to the potential energy via Lagrange multipliers. To prevent numerical difficulties with the stiffness factorization operation, it is essential to augment this constraint with a “penalty” function consisting of functions that are identically zero when equations (14.19) and (14.20) are satisfied. These additional terms do not affect the resulting displacement field. The modified potential function takes the simple form:
where $U$ is the ordinary finite element potential function, $s$ is the penalty scaling constant, and the $\lambda$ are Lagrange multipliers. Contributions to the internal force are computed by taking the variation of equation (14.22), with the following contributions from the constraints in equation (14.22):

$$
\begin{align*}
\Pi &= U(p_1, p_2, \ldots) + \beta \cdot n_2 + \lambda_p \Delta p \cdot n_1 \\
&+ \frac{s}{2} \left[ 1 - (n_1 \cdot n_2)^2 + (\Delta p \cdot n_1)^2 \right] \\
\beta &= \lambda_x \hat{x} + \lambda_y \hat{y}
\end{align*}
$$

(14.22)

The reader will notice that $f_1$ is the force on the first node that is required to constrain the second node to lie on the plane and that there is obviously an equal and opposite reaction $f_2$ at the second node. The moment $m_2$, required to align $n_1$ with $n_2$ is opposed by $m_1$. $m_1$ also has the additional moment $\Delta p \times f_1$ that is generated by the pair of forces $f_1$ and $f_2$. If equations (14.19) and (14.20) are satisfied, equation (14.23) simplifies to (since $n_1 \times n_2$ must also vanish):

$$
\begin{align*}
f_1 &= -(\lambda_p + s \Delta p \cdot n_1) n_1 \\
m_1 &= -m_2 + \Delta p \times f_1 \\
f_2 &= -f_1 \\
m_2 &= -\beta \times n_2 + s n_1 \times n_2 \\
f_{\lambda_x} &= \Delta p \cdot n_1 \\
f_{\lambda_y} &= \hat{x} \cdot n_2 \\
f_{\lambda_y} &= \hat{y} \cdot n_2
\end{align*}
$$

(14.24)

The stiffness (at equilibrium) is symmetric, with the structure
where the vector shows the order of the freedoms. \( \omega \) represents the rotational counterpart of the displacements \( p \) for the node indicated (subscript). The only other nonzero terms are displayed in equations (14.26):

\[
\begin{bmatrix}
    s\mathbf{I}_3 & \mathbf{n}_1^T & -s\mathbf{n}_1^T & 0 & -\mathbf{n}_1 & 0 & 0 & \mathbf{p}_1 \\
    \mathbf{n}_1^T & \mathbf{K}_{12} & -\mathbf{n}_1^T & 0 & -\mathbf{n}_1 & 0 & 0 & \mathbf{p}_2 \\
    -s\mathbf{n}_1^T & \mathbf{n}_1^T & \mathbf{K}_{23} & \mathbf{K}_{24} & \mathbf{n}_1 \times \mathbf{p} & \mathbf{x} \times \mathbf{n}_2 & \mathbf{y} \times \mathbf{n}_2 & \mathbf{p}_2 \\
    0 & \mathbf{K}_{24} & 0 & \mathbf{K}_{44} & 0 & -\mathbf{x} \times \mathbf{n}_2 & -\mathbf{y} \times \mathbf{n}_2 & \mathbf{p}_2 \\
    -\mathbf{n}_1^T & (\mathbf{n}_1 \times \mathbf{p})^T & \mathbf{n}_1^T & 0 & 0 & 0 & 0 & \mathbf{p}_1 \\
    0 & (\mathbf{\hat{x}} \times \mathbf{n}_2)^T & 0 & -\mathbf{\hat{x}} \times \mathbf{n}_2^T & 0 & 0 & 0 & \mathbf{p}_1 \\
    0 & (\mathbf{\hat{y}} \times \mathbf{n}_2)^T & 0 & -\mathbf{\hat{y}} \times \mathbf{n}_2^T & 0 & 0 & 0 & \mathbf{p}_1 \\
\end{bmatrix}
\]

(14.25)

where for any vector \( v \),

\[
\mathbf{\tilde{v}} = \begin{bmatrix}
    0 & -v_z & v_y \\
    v_z & 0 & -v_x \\
    -v_y & v_x & 0
\end{bmatrix}
\]

(14.27)

and where \( \mathbf{I}_3 \) is the 3x3 identity matrix. Note that in deriving equations (14.26), we used the fact that at equilibrium, \( \mathbf{n}_1 = \mathbf{n}_2 \).

It is very simple to derive the noncorotational and linear forms for the force and stiffness by dropping higher order terms, so those details are omitted here.
14.5 Shell and Mesh-Transition Shell Elements

The current version of the STAGS program has the following two E300-series triangular shell elements and the following five E400-series quadrilateral shell and mesh-transition shell elements:

- the E320 triangular shell element
- the E330 triangular shell element
- the E410 4-node quadrilateral shell element
- the E411 4-node quadrilateral shell element
- the E480 9-node quadrilateral shell element
- the E510 5-node quadrilateral mesh-transition shell element
- the E710 7-node quadrilateral mesh-transition shell element

These seven elements are described in the following six Subsections. The final Subsection in this Section describes how mesh transitions can also be performed using triangular elements.
**E320 Triangular shell element**

The **E320** triangular shell element, shown in Figure 14.7, is a “classic” triangular element.

![E320 Triangular shell element](image)

**Figure 14.7** E320 Triangular shell element

The nine integration-point locations are shown here. Element stress/strain results are computed at the element centroid only.
E330 Triangular shell element

The E330 triangular shell element, shown in Figure 14.8, is STAGS’ adaptation of the COMET program’s MIN3 element. * †

In a shell unit, this element may have one, three (the default number), four or seven integration-points—depending on the INTEG parameter on the N-1 record. In an element unit, this element currently has three integration points. Element stress/strain results may be computed at each integration point or just at the element centroid.

Figure 14.8 E330 Triangular shell element

**E410 4–Node quadrilateral shell element**

The **E410** 4–node quadrilateral shell element in the current version of STAGS, shown in Figure 14.9, is a “classic” 4–node quadrilateral shell element.

![Diagram of E410 element]

E411 4–Node quadrilateral shell element

The E411 quadrilateral shell element, shown in Figure 14.10, has four user-specified nodes and an additional set of four mid-side nodes that are constructed by the program.

Figure 14.10  E411 4–Node quadrilateral shell element
**E480 9–Node quadrilateral shell element**

The **E480** 9–node quadrilateral shell element in the current version of **STAGS**, shown in Figure 14.11, is a classic 9–node ANS shell element.* †

![Figure 14.11 E480 9–Node quadrilateral shell element](image)


E510 and E710 Quadrilateral mesh-transition shell elements

Transition meshes can be generated very easily in STAGS shell units with simple input. Two types of juncture lines are recognized, as has been the case for some time. The simplest juncture is defined along an entire shell unit boundary, via G-1 records. The limitation is that only whole boundaries are recognized, and thus it is not possible to insert a unit with double the mesh density along only part of a shell unit boundary, as illustrated in Figure 14.12. To cope with this situation with the minimum number of shell units, the user must join shell units along only part of a juncture line (for example, the line along the bottom of unit 1 in the figure) and place the units properly by using IGLOBE = 0, 3 or 4 (on the M-2 record). This process should be familiar to STAGS users for models containing stiffeners that are modeled as shell units. After the units are properly located, master-slave relationships are generated by G-2 records in which IDIR = 0 for both groups of nodes.

![Figure 14.12 Mesh refinement example](image)

In Figure 14.12, you will also notice the dangling nodes in unit 3. These nodes are omitted from any G-2 reference. To understand how STAGS copes with this situation, it is useful to consider unit 3 in isolation. Each edge or boundary line is numbered clockwise starting with boundary line 1, which runs along row 1; these conventions are also discussed under Record
P-1 (Boundary Conditions). In Figure 14.12, for example, let us assume that row number 1 is the vertical line on the left edge of unit 3 and that column number 1 runs along the bottom side. Then the boundary lines are as labeled in the figure (small circles). Notice that the mesh halves in density along lines 2, 3 and 4, but not along line 1 as one crosses from unit 3 to the adjacent units. Special variables MESH1 through MESH4 (N-1) take the value 0 for a boundary line where the mesh is to be continuous with adjacent shell units, and 1 (or 3) where the mesh halves across the boundary line.

This input signals the code to interpret every other node as a dependent node that is constrained automatically by introducing an E510 quadrilateral mesh-transition element along the boundary for which MESHi is set equal to 1, and/or by introducing an E710 mesh-transition quadrilateral element at any corner for which the MESHi and MESHj parameters for the lines meeting at that corner are both set equal to 1. One can readily see that this has the effect of imposing a halving of the mesh along the affected boundaries. The choice of E510 or E710 depends on the number of dependent nodes that have to be constrained. Note that since the number of intervals is doubling as you move in from an adjacent unit, the number of mesh lines must be 2N-1, where N is the number of mesh lines in the adjoining coarser mesh (the number of mesh lines are specified in the F-1 records). For unit 3 in Figure 14.12, the N-1 record would look like

\[
\begin{array}{cccccccc}
410 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \$
\end{array}
\]

where we have assumed in this example that the user wants to use type E410 elements with unit 3, to have no mesh irregularities, to keep equal mesh spacing, and to use default values for the penalty and integration order. The last four integers correspond to the situation in unit 3, Figure 14.12.

If the user wishes to join the shell units with G-1 records (remembering that the relevant boundary lines must be compatible, as always), the same input applies, except that the input may be simpler since STAGS will often be able to locate the units automatically. In Figure 14.12, unit 3 boundary lines 1 and 3 satisfy G-1 requirements.

In an element unit, quadrilateral mesh-transition shell elements are identified by element type E510 or E710. These elements must be included in the count NQUAD of quads, on record H-1. For either element, if the desired normal is pointing toward the viewer, the 4 corner nodes are numbered first on record T-4a in counter-clockwise order (the same as for the E410); and the remainder of the nodes are input on record T-4b including the midside nodes and the center node (for the E710 only). The midside nodes are numbered in the same order as the corner nodes, starting with the first one to the right (or left for clockwise input) of corner node 1. If there is no midside node along a given side, a 0 is input. This tells STAGS where to put the dependent nodes. It must be remembered that only two topologies are possible, as shown in Figure 14.13. For the E510 case, the single midside node can be located on any of the four sides. The dependent node is opposite the extra node, and is invisible to the user.
For the **E510** element shown in the left half of this figure, the user should input

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 10 \\
0 & 5 & 0 & 0 & 0 & \end{array} \quad \begin{array}{c}
$ T-4a$
\end{array}
\]

\[
\begin{array}{cccccc}
0 & 5 & 0 & 0 & 0 & \end{array} \quad \begin{array}{c}
$ T-4b$
\end{array}
\]

where in this example wall 1 and defaults for the other entries on T-4a are selected. Note that for this case the only nonzero entry on T-4b is in the second slot, since the extra node is located on the second side with reference to node 1, in the counter-clockwise order. The following input describes the **E710** element in the right half of this figure:

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 7 & 10 \\
0 & 5 & 6 & 0 & 7 & \end{array} \quad \begin{array}{c}
$ T-4a$
\end{array}
\]

\[
\begin{array}{cccccc}
0 & 5 & 6 & 0 & 7 & \end{array} \quad \begin{array}{c}
$ T-4b$
\end{array}
\]

Note how similar this input is to the previous example, except that here we have the extra midside node 6 adjacent to 5, and that there is an extra node 7 in the fifth slot. The midside nodes can start on any side and are always adjacent. The center node must appear last as the fifth entry on the record. In a real case, the node numbers will of course correspond to user point numbers generated previously.

For both shell and element unit input, the user will find that just about any type of mesh-doubling scenario can be covered with just these two quadrilateral mesh-transition elements. As viewed from inside the doubled region, the **E510**’s lie along a mesh-transition boundary, and **E710**’s occupy any corners where the mesh is coarser along two adjacent lines. More severe refinement can be done in stages, with one refinement level per boundary line. It will be seen that with the mesh doubling for each line, a very fine mesh can be had in short order.

**Example: Aircraft Panel Quarter Model with Crack in Refined Region**

In this example, we have a quarter of a fuselage panel section containing rings and stringers that are modeled as shell units—a typical ASIP model. Figure 14.14 gives a view of the model and the mesh structure. There are 12 shell units in this example, 7 of which lie on the planform in the figure. Stiffeners, composed of shell units, are marked by the heavy lines. Each shell unit is labeled by a number in the order that it appears in the STAGS input. Arrows point to the stiffener shell units, which are viewed edge-on. All boundaries are symmetry, except for the crack (free edge), marked by the heavy dashed line. The left-hand boundary is symmetry with a specified, constant nonzero $u$. $u$ is aligned along the cylinder axis, which is
horizontal and labeled on the lower left-hand corner of the model. In addition to the specified displacement, a uniform pressure load is applied to units 1 through 7. With each shell unit, we have included the number of rows and columns, rows first, just as they appear on the F-1 record. The direction of increasing rows and column number is also illustrated in the lower left-hand corner of the model.

Now let us walk through the input in this case. You will readily recognize that the F-1 record input matches the numbers in parentheses in Figure 14.14, unit for unit. Check the figure and understand that the mesh numbering corresponds exactly to the number of mesh lines in the row and column direction for each unit. Next come 15 G-1 records that connect whole boundaries between units. Check the figure to ensure that the full boundary line condition is fulfilled. Note that only the first 7 records apply to the skin, while the remainder describe connections to the stiffeners. The next 9 G-2 records are most interesting, because here is a
case where both unit 6 (containing the crack tip) and unit 7 connect to line 2 of unit 4. The first 5 partial compatibility records illustrate the requirement that only *every other* node in the denser-meshed unit 6 is connected to unit 4. As we explained before, the intervening nodes on 6 are *dependent nodes* and are invisible to the user. The next set of 4 compatibility records defines a continuous mesh between unit 4 and 7. The last G-2 record joins the stringer to the ring stiffener.

The first refinement line lies along boundary 4 on unit 3, illustrated on the top half of Figure 14.15, where the unit *boundary lines* are marked by the numerals:

![Figure 14.15 Generation of transition elements](image)

From Figure 14.14, transition elements must be placed along this line to reduce the number of independent nodes to every other line in the unit. As described earlier, the special input on record N-1 tells STAGS where to put dependent nodes. For this reason, we label in the input that boundary line 4 as *reduced* for unit 3. What STAGS actually does is to introduce three E415 quadrilateral elements along line 4, as shown by the shaded squares along line 4. Since unit 4 shows the same pattern as unit 3, it is not surprising that the input to STAGS is identical. Unit 6 is more complicated. Here, doubling occurs along *three* boundary lines (1,
3, and 4 as shown in the lower half of Figure 14.15), with the expected input on the N-1
record. The element pattern shows an \textbf{E417} element in each of the two corners along line 4,
and the remainder \textbf{E415}'s along boundary lines 1, 3, and 4.

The model for this example is constructed \textit{via} the following \textit{INP} file:

\begin{verbatim}
Aircraft Panel Quarter Model—\textit{INP} File

Transition mesh for 15" crack (1/4 model) $ A-1
0 0 1 0 0 0 1 $ B-1
12 0 0 15 10 0 0 0 0 $ B-2
2 0 8 $ B-3
1.0 0.1 1.0 $ C-1
4 9 5 9 7 4 9 4 7 5 9 9 5 5 3 9 3 4 3 5 2 4 2 5 $ F-1
1 3 2 1 $ G-1
1 2 3 4 $ G-1
2 2 4 4 $ G-1
3 3 4 1 $ G-1
3 2 5 4 $ G-1
5 3 6 1 $ G-1
6 3 7 1 $ G-1
1 3 8 3 $ G-1
3 3 9 3 $ G-1
5 3 10 3 $ G-1
8 2 9 4 $ G-1
9 2 10 4 $ G-1
1 2 11 3 $ G-1
2 2 12 3 $ G-1
11 2 12 4 $ G-1
4 1 4 0 6 1 1 0 $ G-2
4 2 4 0 6 3 1 0 $ G-2
4 3 4 0 6 5 1 0 $ G-2
4 4 4 0 6 7 1 0 $ G-2
4 5 4 0 6 9 1 0 $ G-2
4 6 4 0 7 2 1 0 $ G-2
4 7 4 0 7 3 1 0 $ G-2
4 8 4 0 7 4 1 0 $ G-2
4 9 4 0 7 5 1 0 $ G-2
8 2 9 0 11 1 4 0 $ G-2
1 / 10.5E6 0.33 $ I-1/2
2 / 10.7E6 0.33 $ I-1/2
1 1 1 / 1.036 $ K-1/2
2 1 1 / 1.036 $ K-1/2
3 1 1 / 1.036 $ K-1/2
4 1 1 / 1.036 $ K-1/2
5 1 1 / 2.17075 $ K-1/2
6 1 1 / 2.17075 $ K-1/2
7 1 1 / 2.34229 $ K-1/2
8 1 1 / 2.34229 $ K-1/2
$
$ **** SHELL UNIT 1 ****
5 0 $ M-1 #1
0. 10. 165.6 172.8 74.0 $ M-2A #1
\end{verbatim}
1
410 0 0
0 6 6 4
111 100
1/ 1 2
8.5 3 0 0
-.0022 -1 1 1 0
0
$           **** SHELL UNIT 2     ****
$
5 0
10. 20. 165.6 172.8 74.0
2
410 0 0
6 6 4 4
1/ 1 1
8.5 3 0 0
4 0 0 0 0 0
$
$           **** SHELL UNIT 3     ****
$
0 0
0.10. 172.8 174.6 74.
3
410 0 0 0 0 0 0 0 0 0 1
0 6 6 6
111 100
1/ 1 2
8.5 3 0 0
-.0022 -1 1 1 0
4 0 0 0 0 8
$
$           **** SHELL UNIT 4     ****
$
5 0
10. 20. 172.8 174.6 74.
4
410 0 0 0 0 0 0 0 0 0 1
6 6 4 6
1/ 1 1
8.5 3 0 0
4 0 0 0 0 0
$
$           **** SHELL UNIT 5     ****
$
5 0
0.10. 174.6 176.4 74.
3
410
0 4 6 6
111 100
1/ 1 2
8.5 3 0 0
-.0022 -1 1 1 0
0
$
$ **** SHELL UNIT 6  (Contains CRACK tip!) ****

$ M-1 #6  $ M-2A  $ M-5 Wall 4  $ N-1 #6 lines 1,3,4 reduced  $ P-1

10. 15. 174.6 176.4  74. 4
410 0 0 0 0 0 0 1 0 1 1 6 3 6 6
1/ 1 13 8. 5 3 0 0
0. -1 2 2 9 0. -1 4 2 9 0. -1 6 2 9 0. -1 3 9 0. -1 4 3 9 0. -1 6 3 9 0. -1 2 4 9 0. -1 4 4 9 0. -1 6 4 9 0. -1 2 5 9 0. -1 4 5 9 0. -1 6 5 9 0 0 0 0 0 0

$ **** SHELL UNIT 7 ****

$ M-1 #7  $ M-2A  $ M-5 Wall 4  $ N-1 #7  $ P-1

15. 20. 174.6 176.4 74. 4
410 0 0 0 0 0 0 0 0 0 0 6 3 4 6
1/ 1 1 8. 5 3 0 0
0 0 0 0 0 0

$ **** SHELL UNIT 8 ****

$ M-1 #8  $ M-2A #8  $ M-4D #8 (note translation of unit)  $ M-4E #8  $ M-5 #8  $ N-1 #8  $ P-1 #8  $ Q-1  $ R-1 #8

4 4 72.0 74.0 165.6 172.8 10.0 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 3 6 6 4 0
0
0

$ **** SHELL UNIT 9 ****

$ M-1 #9  $ M-2A #9  $ M-4D #9  $ M-4E #9  $ M-5 #9  $ N-1 #9  $ P-1 #9  $ Q-1  $ R-1 #9

4 4 72.0 74.0 172.8 174.6 10.0 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 3 6 6 6 0
0
0
$ **** SHELL UNIT 10 ****

$ 4 4
27.0 74.0 174.6 176.4
10.0 0.0 0.0
0.0 0.0 0.0
6 0 0. 0. 0 0
410 0 0 0 0 0
3 4 6 6 0
0
0

$ **** SHELL UNIT 11 ****

$ 2 3
0.0 1.0 0.0 10.0
0. 9.1493261 -72.424373
10. 9.1493261 -72.424373
10. 9.2746393 -73.416488
7 0 0. 0. 0 0
410 0 0 0 0 0
3 6 6 3 0
0 0 0 0

$ **** SHELL UNIT 12 ****

$ 2 3
0.0 1.0 10.0 20.0
10. 9.1493261 -72.424373
20. 9.1493261 -72.424373
20. 9.2746593 -73.416488
8 0 0. 0. 0 0
410 0 0 0 0 0
3 4 6 6 0
0 0 0 0
0

$
E330 Triangular mesh-transition shell elements

Transition meshes between and among shell units, using type E330 triangular shell elements instead of type E510 and/or E710 quadrilateral elements, can also be generated very easily via the MESH1, MESH2, MESH3 and/or MESH4 parameters (on the N-1 record). This STAGS feature closely parallels the E510 and E710 quadrilateral mesh-transition shell element capabilities described above. Setting MESH1, MESH2, MESH3 and/or MESH4 equal to 3 on an N-1 record signals STAGS to interpret every other node on the associated boundary as a dependent node that is constrained automatically by introducing as many E330 triangular transition elements as may be required along that boundary or at the corner of defined by adjacent boundaries that are tagged in this manner. This has the effect of halving the mesh along the affected boundaries. Since the number of intervals is doubling as you move in from an adjacent unit, the number of mesh lines must be 2N-1, where N is the number of mesh lines in the adjoining coarser mesh (the number of mesh lines are specified in the F-1 records). To employ type E330 (instead of E510 and E710) shell transition elements for unit 3 of the example problem shown in Figure 14.12, the N-1 record would look like

```
410   0   0   0   0   0   0     0 3 3 3  $ N-1
```

(assuming that the analyst wants to use type E410 elements with unit 3, to have no mesh irregularities, to keep equal mesh spacing, and to use default values for the penalty and integration order). The last four integers correspond to the situation in unit 3, Figure 14.12. The difference here is that STAGS generates three type E330 triangular shell elements in each transition “side cell” for which the MESHi value of 1 generates an E510 element, and/or four type E330 triangular elements for each “corner cell” for which adjacent boundaries have MESHi and MESHj both equal to 3. The patterns for this are shown in Figure 14.16:

```
Figure 14.16  Triangular mesh-transition shell elements
```

When the MESH* parameters that were set to 1 in the example just discussed are set to 3, instead of 1, the transition mesh shown in Figure 14.17 is generated:
Figure 14.17  Quarter panel model with crack in refined region
14.6 Sandwich and Mesh-Transition Sandwich Elements

The current version of the STAGS program has the following three “standard” sandwich elements and the following two mesh-transition sandwich elements:

- the E830 6-node sandwich element
- the E840 8-node sandwich element
- the E849 18-node sandwich element

and

- the E845 10-node mesh transition sandwich element
- the E847 14-node mesh transition sandwich element

These are described in the following four Subsections.

**E830 6–Node sandwich element**

Sandwich components play increasingly important roles in the design of many aerospace structures, and it is necessary to determine their behavior adequately. The 6–node sandwich element (see Figure 14.18) discussed here and the 8–node sandwich element (see Figure 14.19) discussed in the next subsection are important additions to STAGS’ modeling capabilities and will be used more and more as time goes on.

The classical approach to the modeling of sandwiches corresponds to an extension of the theory of thin shells. In this approach, the behavior of the sandwich as a three dimensional object is reduced to the behavior of a two dimensional surface with in-plane and out of the plane stiffness properties. Because sandwiches (in contrast to thin walled shells) also undergo transverse shear deformations, two extra parameters are required to describe this mode of deformation. The classical theory of sandwich shells is a two dimensional theory in the sense that the behavior of the three dimensional sandwich shell is described by the deformation of a reference surface and two additional geometric parameters (shear angles) that are locally attached to this surface. Successful finite element discretizations of these classical models have been made in the past, but applications of these discretizations are mostly confined to linear or moderately nonlinear analysis situations.

For implementation into a general purpose finite element code, sandwich elements based on classical theory have at least two undesirable aspects. The first is that a complete new coding of the element drivers must be introduced. The second is that the code must deal with boundary conditions along the edges of the shell that involve shear angles. This departure from standard conventions is cumbersome and requires alterations in the organization of the existing software. A less difficult modeling technique is a welcome alternative.
The classical model of a sandwich reduces the behavior of the aggregate to that of two membranes that are held apart by a core. The core is considered to be virtually inextensional in the transverse direction, usually with a relatively large resistance against transverse shear. However, in the plane of the middle surface of the aggregate, the core possesses virtually no stiffness. It is noted here that it is possible to call a composite of this type a sandwich of the first kind.

A more general type of sandwich emerges if one considers the two faces to consist of shells—with shell surfaces having bending stiffness as well as the usual membrane stiffness—that are held apart by a lightweight core. The latter may have three dimensional elastic properties. Examples of such structural elements are sandwich walls made of glass fibre faces with polyurethane foam as the core material. This type of wall construction can be called a sandwich of the second kind.

The implementation of a sandwich of the second kind in a general purpose finite element code is easier and more straightforward than the implementation of the classical concept of a sandwich in such a code, provided the code possesses a mature shell finite element modeling capability.

**Sandwich element internal force and material and geometric stiffness matrices**

Formulation of the internal force and material and geometric stiffness matrices for a 6–node sandwich element follows standard finite element procedures. An overview of the algorithmic steps used to develop these quantities is outlined in this section. As will be seen, internal force and stiffness matrix terms depend on the element’s constitutive properties and on the components of its displacement gradient field. Detailed discussion of the assumptions made
for the sandwich element’s displacement field, and computation of the displacement field
derivatives, is presented in the next section.

Formulation of the internal force and stiffness matrices begins with the definition of the
engineering Green’s strain, which is given as

\[
\varepsilon = \begin{bmatrix}
  u_x + \frac{1}{2}(u_x^2 + v_x^2 + w_x^2) \\
  v_x + \frac{1}{2}(u_x^2 + v_x^2 + w_x^2) \\
  w_x + \frac{1}{2}(u_x^2 + v_x^2 + w_x^2) \\
  v_y + u_y u_x + v_y v_x + w_y w_x \\
  w_y + v_y u_x + v_y v_x + w_y w_x \\
  u_z + w_z u_x + v_z v_x + w_z w_x
\end{bmatrix}
\] (14.28)

Note that the engineering shearing strains are twice the corresponding tensor values, in
agreement with common practice. Let us assume for simplicity that the internal energy can
be written as the quadratic form

\[
U = \frac{1}{2} \varepsilon^T D \varepsilon = \frac{1}{2} \sigma^T \varepsilon
\] (14.29)

where

\[
\sigma = D \varepsilon
\] (14.30)

and where \( D \) is a constitutive matrix and \( \sigma \) is the stress conjugate to the strain. The
generalization to a nonlinear constitutive function does not affect these arguments.

The first variation of the energy is

\[
\delta U = \sigma^T \delta \varepsilon
\] (14.31)

where \( D \) is assumed to be symmetric and where \( \delta \varepsilon \) is the variation of the strain, which in turn
is a function of the element displacement field through its derivatives in equation (14.28).

In practice, these displacement field derivatives are approximated using so-called
interpolating shape function “formulas.” Without loss of generality, we can define

\[
\begin{bmatrix}
  u_x & u_y & u_z & v_x & v_y & v_z & w_x & w_y & w_z
\end{bmatrix}^T = Gd
\] (14.32)
where the transpose of the row matrix gives the order of the displacement gradient quantities chosen, where $G$ is a matrix of constants (dependent on the initial geometry) with nine rows corresponding to each of the derivatives listed and with as many columns as there are freedoms in the element (in general). $d$ is the vector of nodal displacements (freedoms). It is assumed here that $G$ is known by standard finite element methods. If the derivatives in equation (14.32) are inserted into equation (14.28), with some rearrangement we obtain the variation in the strain:

$$\delta \varepsilon = WG\delta d$$

(14.33)

where the “core strain derivative” matrix $W$ is defined as

$$W(d) = \begin{bmatrix} 1 + u_x & 0 & 0 & v_x & 0 & 0 & w_x & 0 & 0 \\ 0 & u_y & 0 & 0 & 1 + v_y & 0 & 0 & w_y & 0 \\ 0 & 0 & u_z & 0 & 0 & v_z & 0 & 0 & 1 + w_z \\ 0 & u_z & u_y & 0 & v_z & 1 + v_y & 0 & 1 + w_z & w_y \\ u_z & 0 & 1 + u_x & v_z & 0 & v_x & 1 + w_z & 0 & w_x \\ u_y & 1 + u_x & 0 & 1 + v_y & v_x & 0 & w_y & w_x & 0 \end{bmatrix}$$

(14.34)

and where the dependence of $W$ on $d$ is emphasized. Once a set of displacements is known and the derivatives in equation (14.32) are computed, the variation in the strain is known from equation (14.33).

The internal force vector is computed using equation (14.33):

$$f^T \delta d = \sigma^T \delta \varepsilon = \sigma^T WG\delta d$$

(14.35)

from which the force (per integration point) is defined

$$f = B^T \sigma$$

(14.36)

and the nonlinear strain-displacement matrix is given by

$$B = W G$$

(14.37)

The total second variation of the energy as given in equation (14.31) comes in two parts:

$$\Delta \delta U = \Delta \varepsilon^T D \delta \varepsilon + \sigma^T \Delta \delta \varepsilon$$

(14.38)

The first term (referred to as the “material stiffness”) is computed directly from equation (14.38):
The “geometric” second variation always appears as the inner product of the stress with the second variation of the strain:

\[
K_G = G^T \begin{bmatrix} S & 0 & 0 \\ 0 & S & 0 \\ 0 & 0 & S \end{bmatrix} G
\]  

(14.40)

where \( S \) is the 3x3 matrix of stresses in the usual order.

**Sandwich element displacement and displacement gradient calculations**

The displacement of any point in the core of a sandwich element is assumed to vary linearly through the thickness between values at the upper and lower bond lines of the face sheets with the core. If the core’s coordinate system is taken to be the same as that for the lower face sheet reference surface, then the core displacement can be written as

\[
u = a \mathbf{H}_N \mathbf{d}_N + b \mathbf{\hat{H}}_M \mathbf{d}_M
\]  

(14.41)

where \( a = \frac{1}{2}(1 - \zeta) \) and \( b = \frac{1}{2}(1 + \zeta) \) are linear interpolation coefficients parameterized by \( \zeta = \frac{2}{h} z \) and where \( h \) is the core local thickness and \( z \) is the thickness coordinate over the range \([-h/2, h/2]\). The sandwich core shape functions are given by

\[
\mathbf{H}_N = \begin{pmatrix} \mathbf{H}^u_N \\ \mathbf{H}^\theta_N \end{pmatrix}
\]  

(14.42)

\[
\mathbf{\hat{H}}_M = \begin{pmatrix} \mathbf{\hat{H}}^u_M \\ \mathbf{\hat{H}}^\theta_M \end{pmatrix}
\]  

(14.43)

where \( \mathbf{H}_N \) is the matrix of shape functions associated with lower face sheet freedoms at a node \( N \), and \( \mathbf{\hat{H}}_M \) is the matrix of shape functions associated with upper face sheet freedoms at a node \( M \). Both of these matrices are partitioned into two parts that give the contribution to the total displacement due to nodal displacement and nodal rotation freedoms, respectively.

The nodal freedoms for the lower and upper face sheet reference surfaces are also partitioned into displacement and rotation freedoms, respectively:

\[
\mathbf{d}_N = \begin{pmatrix} \mathbf{d}^u_N \\ \mathbf{d}^\theta_N \end{pmatrix}
\]  

(14.44)
The sandwich core displacement given in equation (14.28) allows for the upper face sheet reference surface to be arbitrarily oriented with respect to the lower face sheet reference surface (and the sandwich core coordinate system). If \( x \) and \( \bar{x} \) represent coordinates of a point expressed in lower and upper reference surface coordinate systems, respectively, and are related by an orthogonal rotation matrix \( T \) such that

\[
x = Tx
\]

then the displacement shape function matrices associated with upper face sheet reference surface displacement and rotation freedoms at a node \( M \), but with components expressed in the lower face sheet coordinate system, are given by

\[
\hat{\Phi}_M^u = T \Phi_M^{uT}
\]

\[
\hat{\Phi}_M^\theta = T \Phi_M^{\thetaT}
\]

Here \( \Phi_M^u \) and \( \Phi_M^\theta \) are the displacement shape function matrices associated with upper face sheet displacement and rotation freedoms at a node \( M \), and whose components are expressed in the upper face sheet coordinate system.

The displacement gradient for the sandwich core can now be determined from equation (14.41) to be

\[
\frac{\partial a}{\partial x_i} = \frac{da}{dx_i} \Phi_N d_N + \frac{\partial H_N}{\partial x_i} d_N + \frac{dh}{dx_i} \Phi_M d_M + \frac{\partial \Phi_M}{\partial x_i} d_M
\]

where the gradient of the upper face sheet shape functions with respect to the sandwich core coordinate system is given by

\[
\frac{\partial \hat{\Phi}_M}{\partial x_i} = \left[ \frac{\partial \hat{\Phi}_M^u}{\partial x_i} \quad \frac{\partial \hat{\Phi}_M^\theta}{\partial x_i} \right]
\]

The gradient of the upper face sheet reference surface shape function submatrices identified in equation (14.50) can be expressed in the upper face sheet reference surface coordinate system (the barred system) by using the chain rule along with equation (14.46) and the relations in equations (14.47) and (14.48):
For the sandwich elements that are implemented in the current version of STAGS, the upper and lower face sheets are modeled with shell elements of the same type (i.e., E330 shells for an E830 sandwich and E410 shells for an E840 sandwich). Consequently, using the following definitions for the shape function derivatives,

\[
G_{N_x}^u = \frac{\partial \mathbf{u}}{\partial x} = \frac{\partial \mathbf{u}}{\partial x}, \quad G_{N_y}^u = \frac{\partial \mathbf{u}}{\partial y} = \frac{\partial \mathbf{u}}{\partial y}, \quad G_{N_z}^u = 0
\]  

(14.53)

\[
G_{N_x}^0 = \frac{\partial \mathbf{u}}{\partial x} = \frac{\partial \mathbf{u}}{\partial x}, \quad G_{N_y}^0 = \frac{\partial \mathbf{u}}{\partial y} = \frac{\partial \mathbf{u}}{\partial y}, \quad G_{N_z}^0 = 0
\]  

(14.54)

the components for the sandwich core displacement gradient, as expressed in equation (14.49), can be written as

\[
\frac{\partial \mathbf{u}}{\partial x} = a(G_{N_x}^u G_{N_x}^0)\mathbf{d}_N + b(T(G_{M,x}^u T_{11} + G_{M,y}^u T_{12}) T^T T(G_{M,x}^0 T_{11} + G_{M,y}^0 T_{12}) T^T)\mathbf{d}_M
\]  

(14.55)

\[
\frac{\partial \mathbf{u}}{\partial y} = a(G_{N_y}^u G_{N_y}^0)\mathbf{d}_N + b(T(G_{M,x}^u T_{21} + G_{M,y}^u T_{22}) T^T T(G_{M,x}^0 T_{21} + G_{M,y}^0 T_{22}) T^T)\mathbf{d}_M
\]  

(14.56)

\[
\frac{\partial \mathbf{u}}{\partial z} = \frac{1}{h}(H_N H_N^T)\mathbf{d}_N + \frac{1}{h}(T \mathbf{H}_M T^T T \mathbf{H}_M^T T^T)\mathbf{d}_M + b(T(G_{M,x}^u T_{31} + G_{M,y}^u T_{32}) T^T T(G_{M,x}^0 T_{31} + G_{M,y}^0 T_{32}) T^T)\mathbf{d}_M
\]  

(14.57)

**Displacement at the bond line between sandwich core and face sheet**

Displacement at a bond line between a sandwich face sheet and core is given in terms of the face sheet reference surface displacement and the eccentricity of the bond line with respect to the face sheet reference surface as follows:

\[
\mathbf{u} = \mathbf{u}_R + \Theta \times \mathbf{e}
\]  

(14.58)
Here $u_R$ is the face sheet reference surface displacement, $\Theta$ is the local reference surface rotation, and $e$ is the eccentricity of the sandwich bond line from the face sheet reference surface.

For an $E330$ triangle face sheet shell element, the reference surface rotation components are independent degrees of freedom and are not related to the transverse reference surface $w$-displacement component. In addition, the only nonzero component of the eccentricity is $e_3$. Consequently, the displacement components at a bond line, as given in equation (14.58), can be written as

$$u = u_R + \Theta e_3$$  \hspace{1cm} (14.59)

$$v = v_R - \Theta e_3$$ \hspace{1cm} (14.60)

$$w = w_R$$  \hspace{1cm} (14.61)

From these results, the nodal shape function matrices for the $E830$ sandwich element can be written as

$$H_N^{ij} = \begin{bmatrix} h_N^{u0} & h_N^{u0} & h_N^{u0} \\ h_N^{u0} & h_N^{u0} & h_N^{u0} \\ h_N^{u0} & h_N^{u0} & h_N^{u0} \end{bmatrix}$$  \hspace{1cm} (14.62)

and $h_N^{ij}$ is the shape function for freedom $i$, due to freedom $j$, at node $N$. 

$$H_N^{ij} = \begin{bmatrix} h_N^{u0} & h_N^{u0} & h_N^{u0} \\ h_N^{u0} & h_N^{u0} & h_N^{u0} \\ h_N^{u0} & h_N^{u0} & h_N^{u0} \end{bmatrix}$$  \hspace{1cm} (14.63)
E840 8–Node sandwich element

Formulation of the internal force and material and geometric stiffness matrices for the E840 sandwich element follows exactly the same approach as presented above for the E830 sandwich element. Results for these quantities are given in equations (14.36), (14.39), and (14.40), respectively.

Similarly, the E840 sandwich core displacement follows the same assumptions as presented above for the E830 sandwich element, as represented in equation (14.41). The displacement gradient components are given in equations (14.55), (14.56), and (14.57).

Displacement at a bond line between a sandwich face sheet and core is given in terms of the face sheet reference surface displacement and the eccentricity of the bond line with respect to the face sheet reference surface as follows:

$$ u = u_R + \Theta \times e $$  \hspace{1cm} (14.64)

Here $u_R$ is the face sheet reference surface displacement, $\Theta$ is the local reference surface rotation, and $e$ is the eccentricity of the sandwich bond line from the face sheet reference surface.

For an E410 quadrilateral shell element, Kirchhoff assumptions for the transverse $w$-displacement component at a sandwich bond line can be written as

$$ w = w_R + w_{R,s} e_1 + w_{R,s} e_2 $$  \hspace{1cm} (14.65)
Comparing equation (14.65) with the third component of equation (14.64), the face sheet reference surface rotation vector components are seen to be related to the transverse $w$-derivatives as follows:

\[ \theta_1 = w_{R,y} \] \hspace{1cm} (14.66)
\[ \theta_2 = -w_{R,x} \] \hspace{1cm} (14.67)
\[ \theta_3 = 0 \] \hspace{1cm} (14.68)

In addition, the only nonzero component of the eccentricity is $e_3$. Consequently, the displacement components at a bond line, as given inequation (14.64), can be written as

\[ u = u_R - w_{R,x} e_3 \] \hspace{1cm} (14.69)
\[ v = v_R - w_{R,y} e_3 \] \hspace{1cm} (14.70)
\[ w = w_R \] \hspace{1cm} (14.71)

From these results, the nodal shape function matrices for the E840 sandwich element can be written as

\[
\mathbf{H}^u_N = \begin{bmatrix}
    h_N^{uu} - e_3 h_N^{uw} & h_N^{uw} & h_N^{uw} - e_3 h_N^{ww} \\
    h_N^{vy} - e_3 h_N^{vy} & h_N^{vy} & h_N^{vy} - e_3 h_N^{vw} \\
    h_N^{wy} & h_N^{wy} & h_N^{wy}
\end{bmatrix}
\] \hspace{1cm} (14.72)

\[
\mathbf{H}^\theta_N = \begin{bmatrix}
    \theta_0^{u_1} & \theta_0^{u_2} & \theta_0^{u_3} \\
    \theta_0^{v_1} & \theta_0^{v_2} & \theta_0^{v_3} \\
    \theta_0^{w_1} & \theta_0^{w_2} & \theta_0^{w_3}
\end{bmatrix}
\] \hspace{1cm} (14.73)

and $\theta_N^{ij}$ is the shape function for $u_i$, due to freedom $j$, at node $N$.

E849 18–Node sandwich element

The description of the E849 sandwich element in element units is in preparation.
E845 and E847 mesh-transition sandwich elements

With shell units that are constructed with E840 sandwich elements, transition meshes can be generated in exactly the same way as described above for the standard quadrilateral-shell-elements case. This is done via G-1 and G-2c records (specifying full and partial compatibilities, respectively) and via the MESH1 through MESH4 variables on the N-1 record for any shell unit that has one or more mesh-transition elements. These variables take the value 0 for a boundary line where the mesh is to be continuous with adjacent shell units, or 1 where the mesh halves across the boundary line. This input tells STAGS to interpret every other node on each of the nodal layers of the shell as a dependent node that is constrained automatically by introducing an E845 or an E847 mesh-transition sandwich element along the boundary. The choice of E845 or E847 depends on the number of dependent nodes that have to be constrained. Note that since the number of intervals is doubling as you move in from an adjacent unit, the number of mesh lines in each layer of the shell must be 2N-1, where N is the number of mesh lines in the adjoining coarser mesh (the number of mesh lines are specified in the F-1 records).

The description of 845 and 847 sandwich mesh-transition sandwich elements in element units is in preparation.
14.7 Solid Elements

The current version of the STAGS program has the following four “standard” solid elements:

- **E881** 8-node ANS solid element
- **E882** 18-node ANS solid element
- **E883** 27-node ANS solid element
- **E885** 20-node displacement-based solid element

These are described in the following four subsections.

**E881 8-Node ANS solid element**

The E881 8-node ANS solid element, in STAGS, is shown in Figure 14.20:

![Figure 14.20 The E881 8-node ANS solid element](image-url)
E882 18-Node solid element

The **E882** 18-node ANS solid element, in STAGS, shown in Figure 14.21:

![Figure 14.21  The E882 18-node ANS solid element](image-url)
E883 27-Node solid element

The E883 27-node ANS solid element, in STAGS, shown in Figure 14.22:

Figure 14.22  The E883 27-node ANS solid element
**E885 20-Node displacement-based solid element**

The **E885** 20-node displacement-based solid element, in STAGS, shown in Figure 14.23:

![Diagram of E885 element](image)

**Figure 14.23** The **E885** 20-node displacement-based solid element
14.8 Contact Elements

The three types of contact elements that are implemented in the current version of \textbf{STAGS} are summarized below:

- \textbf{E810} Pad Contact Element
- \textbf{E820} General Contact Element
- \textbf{E822} Line Contact Element

A discussion of each of these element types follows.

\textbf{E810 Pad contact element}

The 8-node surface-to-surface “pad” contact element developed by Carlos Davila\textsuperscript{*} has been implemented in the current version of \textbf{STAGS} as the type \textbf{E810} pad element. This element, shown in Figure 14.24, enables the user to treat some problems in which different shells come into contact with each other or in which shells come into contact with parts of themselves. The pad element is essentially a set of four independent nonlinear springs connecting the corner nodes of two \textbf{E410} shell elements—the “lower” \textbf{E410} coinciding with the lower face and the “upper” \textbf{E410} coinciding with the upper face of the pad. This arrangement facilitates use of a local coordinate system which corotates with the lower \textbf{E410} element, and it provides a simple definition of the average normal to that surface for simple monitoring of the “gaps” that are used to determine the spring forces when contact occurs.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig14.24.png}
\caption{The \textbf{E810} pad contact element}
\end{figure}

\begin{footnotesize}
\begin{itemize}
\end{itemize}
\end{footnotesize}
Forces from the nonlinear corner springs, which enforce the no-penetration inequality constraints on contacting structures, are computed with one or more l-4a tables, which contain stiffness vs. penetration penalty function specifications. The simplest stiffness vs. penetration curve is bilinear—giving zero (or negligibly small) forces and stiffnesses when the gap for a given pair of nodes is zero or positive, and producing large forces and stiffnesses (which act to minimize penetration) when the gap is negative. For best results, the stiffness vs. penetration curves specified in these tables should be “S” shaped—with a pronounced “toe” near the origin. See the “Force and Stiffness Computations” topic, below, for more information about this important subject.

The E810 pad element is designed for use in situations where the contact region is known a priori and where individual elements coming into contact can be readily identified and paired with each other. It enables the user to treat (some) lap-joint and other types of contact problems, for example; but it is not designed for use in more general situations where the contact regions are not known a priori or when sliding occurs or friction is present. Pad elements can also be used to treat contact (minimize penetration) between pairs of 8-node solid elements. They can also be used (in sets of four E810 elements) for contact situations for shells that are constructed with 9-node E480 shell elements. (Note: these capabilities have not been tested and cannot be guaranteed to work correctly with the current version of STAGS.) With the lower pad-element nodes specified as immobilized auxiliary user points, pad elements can be used to treat contact between a flexible shell (the upper surface of the pad element) and a rigid wall (the lower surface of the pad element). This capability has been tested and can be exploited by STAGS users in appropriate situations.

E810 pad elements must be defined in an element unit, referencing User Points (nodes) that coincide with—and are slaved to—the nodes of the underlying E410 shell elements (or E880 solid elements). As noted above, each pad-element definition references a penalty function table that specifies the nonlinear spring stiffness for each of the springs comprising the pad element as a function of penetration.

Force and stiffness computations

As noted above, the 8-node E810 pad element consists of four independent springs connecting the nodes on the lower face of the element to those on its upper face—the lower-face nodes generally being associated with an E410 element on one structural component and the upper-face nodes being associated with an E410 element on a different component. Forces and stiffnesses for the coupled ensemble of springs are expressed in a local element coordinate system which rotates with the lower face element. Neglecting frictional forces, the Davila formulation gives
\[
[F_e] = \begin{bmatrix}
 f^u \\
 f^l
\end{bmatrix}
\]
and
\[
[K_e] = \begin{bmatrix}
 k^u - k^l \\
 -k^l & k^u
\end{bmatrix}
\]  
(14.75)

for the contact-element force vector and stiffness matrix, respectively, with the \( n \) superscript taking on values from 1 to 4 (for the four springs), where

\[
f^n = \begin{bmatrix}
 0 \\
 0 \\
 \gamma^n
\end{bmatrix}
\]  
(14.76)

and

\[
k^n = \begin{bmatrix}
 0 & 0 & 0 \\
 0 & 0 & 0 \\
 0 & 0 & \alpha^n
\end{bmatrix}
\]  
(14.77)

In the Davila formulation, the \( \alpha^n \) quantities are penalty functions representing the spring stiffnesses and the \( \gamma^n = (\alpha^n - \alpha^o)g^n + (U^n - L^n)\alpha^n \) quantities represent spring forces. Here \( \alpha^o \) is the spring stiffness in its “open” configuration, where \( U^n \) and \( L^n \) are the normal displacement components (with respect to the co-rotated lower-face element) of the upper-face and lower-face spring nodes, respectively, and \( g^n \) is the so-called clearance for the nodal pair in the undeformed configuration.

In Davila’s original formulation, \( \alpha^n = \alpha^o \) when \( U^n - L^n - g^n \leq 0 \) and \( \alpha^n = \alpha \) when \( U^n - L^n - g^n > 0 \) (where \( \alpha \) is an analyst-supplied penalty-function parameter for the element). The initial implementation in STAGS of the Davila pad element used these expressions—but with the \( \alpha^n \) defined by the analyst, on an element-by-element basis, as a spring-stiffness-vs-displacement (penalty) function. Numerical difficulties stemming from that implementation were surmounted in the current version of STAGS by replacing the above-cited expression for \( f^n \) by
where \( r^n = r^n(\delta^n) \) is the integrated value of \( \alpha(\delta) \) over \( 0 \leq \delta \leq \delta^n \), and \( \delta^n \) represents the normal penetration of the lower-face reference plane by the \( n^{th} \) upper-face node point.

This approach is especially effective when the user specifies a stiffness-vs-displacement function that, with integration, gives a smoothly increasing force with increasing penetration of the target-element. This approach to force computation for the pad element was motivated by successes realized in overcoming contact-induced numerical problems by using it first with mount elements, and then with the general-contact elements that are described later.

For some examples illustrating some of the ways in which \textbf{E810} PAD elements can be used, please see the \textit{STAGS Elements Manual} and the \textit{STAGS Test Cases Manual} documents.

**E820 General contact element**

The \textbf{E820} point/surface contact “element” was implemented in STAGS to treat structural contact problems that are more general than those that can be handled with the \textbf{E810} element. In discussing this point/surface approach, it is only necessary to consider the case where two structural components experience contact with each other, as shown schematically in Figure 14.25:

![Figure 14.25 Two-body contact situation](image)

One of these bodies is arbitrarily designated as the \textit{contacting} structure, and the other is referred to as the \textit{contacted} structure. These two structures are different bodies in the
following discussion, but everything generalizes to the same- and to the multi-body contact situations.

To use the **E810** (pad) element in an analysis involving structural contact, the analyst must know *a priori* which element(s) of the *contacting* structure come into contact with which elements of the *contacted* structure, so that an 8-node pad element (which is basically a set of four nonlinear springs connecting the corresponding vertices of contacting quadrilaterals) can be constructed for each such pair of elements.

The pad-element approach is inconvenient and inefficient (at best) for problems in which the contacting and/or the contacted elements are not known *a priori*, and it is totally inappropriate for problems in which the contacting and/or contacted elements change as the analysis progresses. More general *surface/surface* or *point/surface* contact capabilities are required for problems of that nature.

With a completely general surface/surface contact capability, the analyst would only need to specify that two (or more) structures might experience contact with each other, and the program would do whatever it needs to detect and deal with contacts as they occur. It is something of an understatement to note that this would be a very expensive luxury for most problems of significant size and/or complexity. In many problems, good advantage can be taken of user specifications that some part(s) of the contacting structure may experience contact with some part(s) of the contacted structure, so that only elements and/or points in the designated regions need to be considered during the contact operations that the program must perform.

With this level of generality, a *surface/surface* contact implementation permits the analyst to specify that one or more members of a specific set of elements on the surface of the contacting structure may initially be in contact (or may subsequently come into contact) with one or more members of a different set of elements on the surface the contacted structure. The program would then need to detect and deal with all kinds of situations in which various parts of each contacting element interact with various parts of one or more contacted elements. That is generally computationally demanding and very expensive.

With the same level of generality, a *point/surface* implementation permits the analyst to specify that one or more contact points on the surface of the contacting structure may initially be in contact (or may come into contact) with one or more members of a set of target elements on the surface the contacted structure. Here, the program need only detect and deal with interactions between individual contact points and the specific target elements that comprise the contact surface.

This point/surface approach has been implemented in **STAGS**, to treat the situation that is shown schematically in Figure 14.26:
For problems requiring this more general treatment of structural contact, the user of STAGS must include one or more so-called contact-definition specifications in an element unit for the model to be analyzed. With each contact-definition, the analyst specifies that one or more contact points in the contacting structure may experience contact with one or more of the target elements that comprise the contact surface region of the contacted structure. (The contact points and the contacted surfaces are usually, but not necessarily, in different shell units in the model.) STAGS then considers the possibility that during the analysis any of these contact points may be in contact with any of these target elements—where each point in contact with the contact surface can move (without friction) from one target element of the contact surface to a neighbor, as loadings and deformations dictate.

It is important to note that each point/surface contact definition produces one E820 “contact-point element” in the STAGS model for each designated contact point, and one E821 “target element” in the model for each designated target element. No actual structural elements are defined by these definitions.

The analyst informs STAGS (via contact definitions) at the outset of an analysis that the indicated contacts are possible. STAGS then takes contact-induced forces and stiffnesses into account during the course of the analysis by using appropriate combinations of these E820 and E821 elements to construct structural elements on-the-fly when they are required for each actual contact between a contact point and the contacted target element.

Two major aspects of the current point/surface contact implementation in STAGS are discussed in the following two sub-sections: how STAGS determines the status of each contact point—i.e., how the program determines whether or not each contact point is in contact with its target surface (and, if so, which element(s) it is contacting), and how STAGS computes the

---

Figure 14.26  Point/surface contact definition

<table>
<thead>
<tr>
<th>Contacting Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contact Point</td>
</tr>
<tr>
<td>Contact Surface</td>
</tr>
<tr>
<td>Target Element</td>
</tr>
<tr>
<td>Contacted Structure</td>
</tr>
</tbody>
</table>
contact-induced forces and stiffnesses. Results obtained using the new point/surface contact capabilities are presented and discussed after that.

**Determination of contact status**

Information about contact is organized on two distinct levels in STAGS. On the first level, analyst-supplied contact-definition information is used to generate a number of multi-value dictionaries (tables) that are used to keep track of which contact-points and target elements are included in each contact-definition. These tables facilitate access to information such as (a) the node points that are associated with the set of target elements for the contact region, (b) the target elements that are attached to each of these nodes, (c) the other target elements (if any) that are connected to each edge of each target element on the target surface, and other things like that. On the second level, contact status information is maintained for each contact point, and configuration information is maintained for each contact point and each target element. Some of this information is maintained in datasets that disappear on completion of the analysis, and some of it is maintained in the case.rst file along with other information required to re-start a STAGS analysis.

When STAGS performs a first or second variation for a problem, the program checks first to determine the contact status of each contact point. Figure 14.27 shows an overview of the logic used in STAGS to determine the contact-status.

![Flowchart showing contact-status determination](image)

**Figure 14.27** Contact-status determination

If the contact point is not in contact with its contact surface, STAGS uses the logic shown in Figure 14.28 to determine whether or not that point initiates contact with its contact surface during the current step.
STAGS begins by comparing the current position of the point to that of each node on the contact surface to identify \textit{NABOR}—the contact surface node closest to the point. STAGS then examines the positional relationships between point and each target element attached to \textit{NABOR} at the start of the current step and for the current solution, to determine if point is on the same side of the element as it was at the start of the current step (in which case contact is not initiated), or if the path of point (during the current step) has crossed the element (in which case contact is initiated). STAGS takes the target element thicknesses into account, and utilizes a user-specified contact-point \textit{RADIUS} parameter to account for the thickness of the contacting structure, in determining whether or not contact occurs. If the path of this contact point crosses the interior of a given target element, the point initiates contact only with that particular target element. If the crossing occurs on an edge or at a vertex of the target element, the point initiates contact with the crossed target element and with its neighbor (on the crossed edge) or with its neighbors (at the crossed vertex).

The algorithm that STAGS uses to determine initial contact also has an exhaustive-search option, to examine all contact-point/target-element contact possibilities (instead of just those stemming from \textit{NABOR}). This option is in the code but cannot be user-activated at the present time. If the contact point is in contact with its contact surface, STAGS uses the logic shown in Figure 14.29 to determine whether or not that contact continues.

Subroutines \textbf{E820Q2} and \textbf{E820T2} (in the \textit{No} branch from the “\textit{Is the contact point inside the element at start of step?”} question, in Figure 14.29) are called when the target element that the contact point is contacting at the start of the step is a quadrilateral or a triangle,
respectively. Each of these routines tests to determine (a) if during the current step the contact point crosses the surface of the target element within its domain (in which case contact with the entire contact surface is terminated), (b) if the contact point is still within the interior of the start-of-step target element (in which case contact with that target element continues), and/or (c) if the contact point is currently on an edge or at a vertex of the target element (in which case contact continues with all of the target elements that are attached to that vertex. If it is determined that the contact point continues its contact with the given target element (or that it stops contacting the entire contact surface), an “I am finished!” signal is returned to the calling routine (along with the necessary contact-point status parameters).

The “Examine exit-edge neighbor element” and “Examine other target segment(s) ... at the exit vertex” portions of the Yes branch from the “Is the contact point inside the element at start of step?” question, in Figure 14.29) are basically calls to control routines (NABOR1 and AMIGO1) that perform similar tests for the contact point with respect to the one target element that is connected to the flagged edge of the original target element or to each of the target elements that are connected to the flagged vertex of the original target element, respectively. An “I am finished!” signal from either of these routines terminates efforts to determine if contact continues, with the contact-point status fully determined. If either routine returns to the control level (shown in Figure 14.29) without the “I am finished!” signal, testing continues until all viable contact-continuation candidates have been checked.

Here, it is important to note that continued testing may be required for three reasons: because the path of the contact point, originally in the interior of a given target element, has taken it across an edge of or through a vertex of that element (then into or through a neighboring element, if any); because the point, originally on an edge of the given target element, has
moved off that edge (and possibly into or through the neighboring element, if any); or because the point, originally at a vertex of the given target element, has moved off that vertex (and possibly into or through a neighboring element, if any). The word *through* is used in two senses in the preceding sentence: some tests are performed to determine whether or not the path of the contact point crosses the appropriate surface plane of the candidate element (in which case contact ceases), and other tests are performed to determine if the path of the contact point is such that the candidate element is traversed without crossing the surface of that element (in which case contact *may* continue).

Under some conditions, the contact status of a given contact point and its target surface cannot be determined without adding a few “twists” to these tests, principally in order to take more global (than at the element level) contact-surface curvatures into account. In essence, these twists supplement the simple “*Am I in the domain of the (given) element?*” test that *STAGS* uses throughout most of its contact-status determination process.

**Force and stiffness computations**

The general-contact quadrilateral element used in *STAGS* is based on the variational approach used by Parisch. The general-contact triangular element in *STAGS* is based on the formulation that is described farther below.

The Parisch formulation for the contact stiffness matrix and force vector begins with the assumption that a contact point is penetrating a target element at a known location, \( C(\xi, \eta) \), where \( \xi \) and \( \eta \) are the *surface coordinates* of the contact point. Applying the usual variational techniques to the energy functionals, Parisch obtained the following consistent tangent stiffness matrix and force vector for a quadrilateral target element:

\[
K_e = \varepsilon\{\Theta_1 + \Theta_2 + \Theta_3 + \Theta_4 + \Theta_5\}
\]  
(14.79)

and

\[
F_e = \varepsilon\delta \cdot N
\]  
(14.80)

where

\[
\Theta_1 = N \cdot N^\ast
\]  
(14.81)

In these expressions, the $\psi_n$ are the basis functions evaluated at the nodes of the target element; $e^1$, $e^2$, and $n$ are the contravariant tangent vectors and the normal vector at the contact point on the target element; $h$ and the $m_{ij}$ are contact-point-related geometric quantities; $\delta^3$ measures warping of the deformed target element surface; $\delta$ represents the penetration by the contact point of the target element at $c(\xi,\eta)$; and $\epsilon$ is the user-supplied penalty function parameter.

For the triangular element, the normal-direction components of the material stiffness matrix are

$$[K_m] = \alpha \begin{bmatrix} L_1L_1 & L_1L_2 & L_1L_3 & -L_1 \\ L_2L_2 & L_2L_3 & -L_2 \\ L_3L_3 & -L_3 \\ \text{sym} & 1 \end{bmatrix}$$ (14.91)

and the full geometric stiffness matrix is

$$[K_g] = K_{g_1} + K_{g_2} + K_{g_3} + K_{g_4}$$ (14.92)

where
where

\[ P^* = \begin{bmatrix} 0 & h & 0 & 0 & -h & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & (b - s) & 0 & 0 & s & 0 & 0 & -b & 0 & 0 \end{bmatrix} \]  
\[ (14.97) \]

\[ Q^* = \begin{bmatrix} h & (b - s) & 0 & -h & s & 0 & 0 & -b & 0 & 0 & 0 \end{bmatrix} \]  
\[ (14.98) \]

\[ R^* = \begin{bmatrix} 0 & 0 & r_1 & 0 & 0 & r_2 & 0 & 0 & r_3 & 0 & 0 \end{bmatrix} \]  
\[ (14.99) \]

\[ S^* = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \]  
\[ (14.100) \]

\[ T = \begin{bmatrix} 0 & 0 & 0 & 0 & -x_2 & 0 & 0 & x_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & 0 & 0 & -x_1 & 0 & 0 & 0 & 0 \\ 0 & x_2 & -x_1 & 0 & -x_2 & x_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -x_2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & x_2 & -x_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ S & y & m & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \]  
\[ (14.101) \]

where \( b \) and \( h \) are the base and height dimensions of the triangle (in its local coordinate system); the \( L_i \) are the basis functions (area coordinates) in the triangle at the point of contact; \( \delta \) is the penetration; \( s \) is the (local-system) x-coordinate of the triangle’s third (non-base) node point; and where the \( r_i \) and \( x_i \) parameters are functions of the contact-point location in the triangle. The \( \alpha \) and \( f \) parameters, here, represent the user-specified stiffness (as a function of the penetration) and its integration over the stiffness/penetration curve, as described above.
The normal-direction force contributions for the triangular element are given by

\[ \{F\} = \alpha \cdot \begin{bmatrix} -L_1 \\ -L_2 \\ -L_3 \\ 1 \end{bmatrix} \]  

(14.102)

For examples illustrating the use of E820 general contact elements, see the STAGS Elements Manual and the STAGS Test Cases Manual documents.

**E822 Line contact element**

The line contact element that STAGS constructs on-the-fly when contact occurs between two line-pair segments is based on the formulation that is described in the paper by Rankin, Chien, Loden and Swenson.*

Let us suppose that two lines in space are defined by ordered sets of nodes \{N_i\} and \{M_i\} for contacting line L1 and contacted line L2, respectively, as shown in Figure 14.30. Also let us assume that any point \( p \) between any two nodes \( N_i \) and \( N_{i+1} \) on a given line is described by linear interpolation.


![Figure 14.30 Line-contact lines](image-url)
We have labeled the minimum distance $d$ between the two lines, as shown in the figure. Any point between, for example, two nodes $N_i$ and $N_{i+1}$ is interpolated by a parameter $\xi$ as follows:

$$p(\xi) = \frac{1}{2} [p_1(1-\xi) + p_2(1+\xi)]$$  \hspace{1cm} (14.103)

and for the second line, we have

$$q(\eta) = \frac{1}{2} [q_1(1-\eta) + q_2(1+\eta)]$$  \hspace{1cm} (14.104)

where this time, the points $q$ refer to nodes $M_i$ and $M_{i+1}$ on the other line. The distance between $p$ and $q$ is clearly

$$d(\xi, \eta) = p - q$$  \hspace{1cm} (14.105)

To determine if contact has taken place, we wish to find the minimum value of $d$ as a function of $\xi$ and $\eta$, or

$$d \cdot p,\xi = 0$$

$$d \cdot q,\eta = 0$$  \hspace{1cm} (14.106)

Since

$$p,\xi = \frac{1}{2} (p_2 - p_1)$$

$$q,\eta = \frac{1}{2} (q_2 - q_1)$$  \hspace{1cm} (14.107)

Equations (14.107) yield a linear equation in $\xi$ and $\eta$:

$$\begin{bmatrix} |p_2 - p_1|^2 & -(p_2 - p_1) \cdot (q_2 - q_1) \\ -(p_2 - p_1) \cdot (q_2 - q_1) & |q_2 - q_1|^2 \end{bmatrix} \begin{bmatrix} \xi \\ \eta \end{bmatrix} = \begin{bmatrix} -2\Delta d_c \cdot (p_2 - p_1) \\ 2\Delta d_c \cdot (q_2 - q_1) \end{bmatrix}$$  \hspace{1cm} (14.108)

or

$$Ce = r$$  \hspace{1cm} (14.109)
where $\Delta d_c$ is the distance between the centroids of the line segments belonging to the nodes in question:

$$
\Delta d_c = \frac{1}{2}[ (p_2 + p_1) - (q_2 + q_1) ]
$$

(14.110)

and $C$, $e$, and $r$ in equation (14.109) correspond to the matrix and vectors in equation (14.108). A solution to equation (14.108) always exists except in the pathological case that the two lines are parallel to one another, in which case an infinite number of solutions exist; such a case can be treated by other methods. A suggested technique is to find which pair of centroids yields a minimum distance, and then solve equation (14.108). If equation (14.108) has a solution in the range $[-1, +1]$ for both $\xi$ and $\eta$, the segment with minimum distance has been found; otherwise, neighboring segments need to be tried.

The penalty method of contact enforcement is expressed through the enforcement function $g(|d|)$ which acts to repel the segments whenever $d$ is negative, negative being defined when any segment crosses any shell or solid element face. To determine whether a crossing has occurred, combine equation (14.103) with the equation interpolating the element surface, and solve. Taking the corner nodes and using a bilinear interpolation:

$$
\frac{1}{2}[ p_1(1 - \xi) + p_2(1 + \xi) ] = \frac{1}{4}\sum_k(1 - rr_k)(1 - ss_k)x_k
$$

(14.111)

for the unknowns $\xi$, $r$, and $s$. If there is a crossing, all three of these variables will lie in the interval $[-1,1]$. If $d$ is negative, then the segments will be repelled by a force proportional to its first variation:

$$
\delta g(|d|) = \frac{g'}{|d|} d \cdot \delta d = \hat{d} \cdot \delta d
$$

(14.112)

where $g'$ is the derivative of $g$ (or the quantity $f$ most often interpolated from a one-dimensional table) and $\hat{d} = d/|d|$ is the normalized minimum distance vector. Expressed in terms of the problem unknowns, we obtain

$$
\delta g(|d|) = \frac{1}{2}g' \hat{d} \cdot [(\delta p_2 + \delta p_1) - (\delta q_2 + \delta q_1) - (p_2 - p_1)\delta \xi - (q_2 - q_1)\delta \eta + \xi(\delta q_2 - \delta q_1) - \eta(\delta q_2 - \delta q_1)]
$$

$$
= \frac{1}{2}g' \hat{d} \cdot [(\delta p_2 + \delta p_1) - (\delta q_2 + \delta q_1) + \xi(\delta p_2 - \delta p_1) - \eta(\delta q_2 - \delta q_1)]
$$

(14.113)

where dependence on $\delta \xi, \delta \eta$ vanishes because of equations (14.106) and (14.107).

The first and second variation is simplified by a straightforward change of variables:
The “internal force” due to contact becomes

\[ \mathbf{p} = v_1 + \xi v_3 \]
\[ \mathbf{q} = v_2 + \eta v_4 \]
\[ \mathbf{d} = v_1 - v_2 + \xi v_3 - \eta v_4 \]  

The “internal force” due to contact becomes

\[
\begin{bmatrix}
I \\
-I \\
\xi I \\
-\eta I
\end{bmatrix}
\begin{bmatrix}
f' \\
-f' \\
f''
\end{bmatrix}
= g' \mathbf{Z} \hat{\mathbf{d}}
\]

where the matrix \( \mathbf{Z} \) will appear many times. The force transforms to the original system by multiplying by the transpose of \( \mathbf{R} \).

The second variation comes in three parts. The so-called “material” portion in the modified coordinate system can be written down immediately:

\[
\mathbf{K}_m = \mathbf{Z} \hat{g}''(\hat{\mathbf{d}})^T
\]

where \( g'' \) is the second derivative of \( g \), or the first derivative \( f' \) of the table quantity \( f \).

Although more difficult to compute, the “geometric” part is made much simpler by our change of variables. The first task is to determine the derivatives of \( \xi \) and \( \eta \) with respect to \( \mathbf{v} \), where we leave off the subscript on \( \mathbf{v} \) for now. To do this, we first note that equation (14.109) is transformed into the new system as follows:
Now we take the derivative of equation (14.118):

\[
\ddot{\mathbf{C}}_v \dot{\mathbf{e}} + \ddot{\mathbf{C}}_{e,v} = \ddot{\mathbf{r}}_{,v}
\]  

(14.119)

from which the unknown quantities \( e_{,v} \) can be computed. The following derivations are lengthy but straightforward:

\[
\ddot{\mathbf{C}}_{e,v} = \ddot{\mathbf{r}}_{,v} - \ddot{\mathbf{C}}_{,v} \dot{\mathbf{e}}
\]

\[
\begin{bmatrix}
-\mathbf{v}_3^T & \mathbf{v}_3^T \\
-\mathbf{v}_4^T & \mathbf{v}_4^T
\end{bmatrix}
\begin{bmatrix}
\mathbf{v}_3 \\
\mathbf{v}_4
\end{bmatrix}
\begin{bmatrix}
\mathbf{v}_1 - \mathbf{v}_2 + 2\mathbf{\xi}_v \mathbf{v}_3 - \mathbf{\eta}_v \mathbf{v}_4 \\
\mathbf{\xi}_v \mathbf{v}_4 - \mathbf{\eta}_v \mathbf{v}_3
\end{bmatrix}
\]

(14.120)

Note that there are 12 columns on the right hand side of equation (14.120), which correspond to the derivatives with respect to each component in turn of the transformed independent variable \( \mathbf{v} \). The result is a 2 x 12 matrix of derivatives \( e_{,v} \).

Next, we must compute the derivative of the unit vector \( \dot{\mathbf{d}} \) by the well-known formula

\[
\ddot{\mathbf{d}}_{,v} = \left( \mathbf{I} - \dot{\mathbf{d}} \dot{\mathbf{d}}^T \right) \frac{\mathbf{d}_{,v}}{|\mathbf{d}|}
\]

\[
= \mathbf{Q} \frac{\mathbf{d}_{,v}}{|\mathbf{d}|}
\]

(14.121)

\( \mathbf{d}_{,v} \) is computed long-hand to yield:

\[
\mathbf{d}_{,v} = \left[ \begin{bmatrix} \mathbf{I} + \mathbf{v}_3 \mathbf{\xi}, \mathbf{v}_1 - \mathbf{v}_4 \mathbf{\eta}, \mathbf{v}_1 - \mathbf{I} + \mathbf{v}_3 \mathbf{\xi}, \mathbf{v}_2 - \mathbf{v}_4 \mathbf{\eta}, \mathbf{v}_2 \mathbf{\xi} \mathbf{I} + \mathbf{v}_3 \mathbf{\xi}, \mathbf{v}_3 - \mathbf{v}_4 \mathbf{\eta}, \mathbf{v}_3 - \mathbf{\eta} \mathbf{I} + \mathbf{v}_3 \mathbf{\xi}, \mathbf{v}_4 - \mathbf{v}_4 \mathbf{\eta}, \mathbf{v}_4 \end{bmatrix} \right] \left[ \mathbf{v}_3 - \mathbf{v}_4 \right] e_{,v}
\]

(14.122)
If we now combine equations (14.121), (14.122), and use the chain rule on equation (14.116), we obtain the following:

\[ K_g = K_1 + K_2 \]

\[ K_1 = g'ZQd_{,v} \]

\[ K_2 = g' \begin{bmatrix} 0_{12} \\ 0_{12} \end{bmatrix} \]

\[ K_2 = g' \begin{bmatrix} \hat{d} & 0_3 \\ 0_3 & e_{,v} \end{bmatrix} \]

Note that for equations (14.123), \( 0_{12} \) represents a 3x12 matrix of zeros and \( 0_3 \) a 3x1 matrix of zeros. Also, recall that dimensions of \( e_{,v} \) and \( d_{,v} \) are 2x12 and 3x12 respectively, as they must be.

Finally, the total stiffness (second variation) is

\[ K_{tot} = K_m + K_g \]

(14.124)

and that the stiffness in the original system is

\[ K_{tan} = R^T K_{tot} R \]

(14.125)

where \( K_{tan} \) is the required output “tangent” stiffness for the contact penalty function with freedoms arranged as in equations (14.103) and (14.104). That transformation can be carried out symbolically, resulting in the following:

\[ K_{tan} = R^T \begin{bmatrix} K_{11} & K_{12} & K_{13} & K_{14} \\ K_{12} & K_{22} & K_{23} & K_{24} \\ K_{13} & K_{23} & K_{33} & K_{34} \\ K_{14} & K_{24} & K_{34} & K_{44} \end{bmatrix} R \]

(14.126)
See the *STAGS Elements Manual* and the *STAGS Test Cases Manual* documents for examples illustrating the use of E822 line contact capabilities.
15 Analysis Techniques

15.1 Modeling Strategy

The modern computer has made it possible for the structural analyst to consider models that approximate rather closely the actual structure. However, there are still several limitations set by computer runtime and storage capacity. The quality of the analysis that can be achieved within a fixed budget depends largely on the ability of the analyst to transform the given structure into a tractable model. In cases requiring a significant computer expense, the user with moderate experience would be well advised to consult with an expert on such questions as modeling, choice of analysis method, and strategy used in the nonlinear analysis. The simplifications that can be made and the possible pitfalls in modeling are so diverse that it is impossible to give advice that covers all significant questions. The interested reader should study the Bushnell papers given in the footnote.*

A general principle that almost always should be followed is to consider carefully before the analysis what kind of results are to be expected. Unexpected developments tend to result in a need to rerun the analysis after modification of the model. When the analysis is planned it is prudent to assume that more than one run will be needed even without allowance for redesign. Questions about modeling of the structure can probably best be answered after some preliminary analyses of models with relatively few degrees of freedom.

  and
  and
If the budget situation does not allow a nonlinear analysis of a relatively true model of the structure, a bifurcation (linear pre-buckling) analysis may be used instead. It may be possible to verify the adequacy of such an approach by comparison of results from nonlinear and bifurcation analyses for a smaller model that can be expected to exhibit similar behavior. Another possibility is to make a nonlinear analysis of only a part of the structure. For example, for a shell with several cutouts the influence of each cutout may be evaluated separately. If the cutouts are sufficiently far apart, the interaction of their effects may be disregarded. It may be possible to conclude that the effects of geometric nonlinearity in some part of the structure are negligible. Such a conclusion may for example be based on a preliminary bifurcation buckling analysis. In that case, it is possible to omit the nonlinear terms in one or more branches [see the description of the M-1 record, in Chapter 6].

It is sometimes necessary to model the structure as composed of several shell units; in other cases, it may be convenient but not necessary to do so. It may be possible to avoid use of a user written geometry subroutine by definition of more than one shell unit. This affects the numbering system for the grid points, however, and may therefore result in increased computer time.

When a structure is made up of several shell units, the unknowns are in general assigned freedom numbers in the order in which the shell units are given. The way in which the units are numbered can significantly affect the computational effort and disc storage required for assembly and decomposition of the overall stiffness matrix. The greatest efficiency is obtained by minimizing the differences between freedom numbers on boundary lines which are connected. Thus the user should attempt to define his structure so that successive shell units are connected where possible with the last row or column of one unit connected to the first row or column of the next unit.

Any one of the shell units must be defined so that it can be mapped onto a surface with four edges. For this purpose, it is sufficient to define a numbering system by rows and columns for the grid points. An example is shown in Figure 15.1, where a shell surface (a quarter of a plate with a rectangular hole) with five “edges” is mapped onto a rectangular surface. Triangular domains approximated with finite elements can be introduced to make the modeling of the shell easier.

A 360° segment of a shell of revolution (or any “two–sided” shell unit) can be mapped onto a four–sided domain if some generator is chosen to represent both sides 2 and 4. A data record is included to indicate displacement compatibility on these two sides. If the shell of revolution defined by user-written subroutines is closed at the apex, the user defines a shell with a hole at the apex. In addition, regular data records can be used to define triangular elements at the apex. For standard shell geometries, the STAGS program automatically includes a number of triangular elements.
Sometimes it is difficult to decide whether a part of the structure, such as a relatively deep ring, should be represented as a separate shell unit or as a stiffener. If it is modeled as a stiffener, it follows that the effect of warping and cross-section distortions are omitted. Clearly, this may lead to unconservative results. For example, the T-stiffener shown in Figure 15.2 has the additional freedom represented by bending of the web. The flange will rotate through an angle which is smaller than the rotations occurring during buckling. A rather extensive study of this general problem is presented in a classic paper by D. Bushnell. Cross-section deformation is also the reason that a reduction factor on the torsional stiffness must be introduced for the corrugation stiffened shell [see the description of the K-4b record, in Chapter 5]. The results presented in the above–cited Bushnell paper may be of some help in the choice of the constant PHI (K-4b). Structural elements are defined as stiffeners if the strain energy can be sufficiently accurately determined in terms of the value of the displacement components at the reference surface. The presence of a stiffener does not add much to the number of degrees of freedom and considerable computer time may be saved by modeling a part of the structure as a stiffener. Sometimes, if use of the stiffener approach is too unconservative, stiffeners with reduced stiffness properties may be defined rather than one or more extra shell branches. Separate studies on small models may be performed to determine such reduction factors.

The analysis of a stiffener is based on beam theory. Contributions to the final equations from beams and stiffeners are computed in the same parts of the computer program. They are handled separately in the input because a stiffener always has displacements that are compatible with those of a shell unit. It is assumed in the beam theory that the cross-section of a beam remains plane after deformation. Therefore, unless thermal expansion or inelastic strains complicate the situation strain energy can be defined in terms of reference surface quantities and a few geometric parameters—such as area and moments of inertia. For that case the beam can be defined as having a “cross-section of general type.” However, the user might find it more convenient to define the cross-section as consisting of a number of rectangular sub-elements.

If the beam is subject to bending, plastic strains will not vary linearly with the distance to the reference surface. In that case it is not sufficient to define the geometric parameters of the cross-section. It is necessary to describe the cross-section in detail. In STAGS that is done by breaking the cross-section up into a number of elements in such a way that the moment of inertia of each sub-element can be ignored. The same method is used if the temperature varies over the stiffener cross-section. A cross-section of sub-element type can be defined as consisting of only rectangular sub-elements. In that case the user gives the dimensions of the element and its area and moments of inertia are computed. This method often represents the easiest way to define a cross-section composed of rectangular parts.

It is possible to consider the stiffeners as discrete. In that case the location of each stiffener must be defined. The other option is to define the stiffeners as “smeared,” that is, their
contribution to the shell wall stiffness is considered to be uniformly distributed over the shell surface. The shell stiffener combination is then represented by an equivalent orthotropic shell wall.

Use of discrete stiffeners entails more work in preparation of data records. Also, it requires some additional computational effort and computer storage space. Discrete stiffeners must be used if the possibility of lateral buckling of the stiffeners is considered and if the shell wall deformation between stiffeners is important. Notice that then it is also necessary to include a few points between the stiffeners. Therefore, a model in which there are no grid lines between discrete stiffeners would often have been better defined with smeared stiffeners. An exception may be the case in which the cross-section of the stiffeners varies from one stiffener to another but not along the length of the stiffeners. In that case the definition of smeared stiffeners in a user-written WALL will entail as much work as the discrete stiffener option. Smeared stiffeners may be used to suppress the shell wall deformation between stiffeners.

The position of a stiffener that does not follow a grid line is defined by use of a functional relationship in a user-written subroutine.

Standard boundary conditions (applied to an entire boundary line) may be specified by Boundary Condition data records. Point-wise constraints may be specified by load records (Q-1, Q-2) or in user written subroutine USRLD. Any other linear displacement constraints can be specified by user-written subroutine UCONST.

If a free body is subjected to a self-equilibrated load system, it is advisable, and sometimes necessary, to add artificial constraints to prevent rigid body motions. These of course, must be chosen so that they do not prevent deformation of the shell. In that case, zero forces are exerted on the shell at the points of artificial constraint.

In order to make sure that no rigid body motions are permitted, consider the structure in any Cartesian coordinate system that is convenient in the case. Make sure that translation along and the rotation about each of the three coordinate axes are constrained. Notice, for example, that a cylindrical shell “simply supported” at both ends is free to move in the axial direction. Such motion is prevented by restraining (Q-1, Q-2) axial motion at one point on the shell surface.
15.2 **Solution Strategy**

The first task, confronting the analyst of a shell structure, is to consider possible failure modes and based on this to decide what type of analysis he needs to perform. A nonlinear transient analysis with an accurately determined loading history answers all question regarding structural adequacy, barring the special conceptual difficulties in connection with inelastic deformation and imperfection sensitivity. However, such an analysis is almost as a rule too expensive and the analyst must consider the possibilities of obtaining approximate solutions that still are satisfactory.

*Is the problem static or dynamic?*

A static analysis serves as a satisfactory approximation if the load is applied sufficiently slowly. It may also be accepted if can be shown to be conservative.

If the expected mode of failure is due to local excessive stresses (brittle failure, spallation) rather than a general collapse of the structure (buckling), inclusion of dynamic effects (stress wave propagation) may be required if the time for application of the load is small in comparison to a typical distance (length, radius) divided by the velocity of sound in the material.

If the expected mode of failure is collapse of the structure or critical stresses (brittle failure or excessive plastic strain) primarily due to long wave bending stresses, the time for load application should rather be compared to the period for the vibration modes that are dominant in the deformation pattern causing collapse. If the time it takes to apply the load is about the same or larger than the time for a period of the slowest vibration mode, the static analysis gives satisfactory results. Not much experience is available in this area, but this contention seems to be corroborated by results presented in 1974 by Almroth and Meller.* It should be noticed that the period of vibration varies with the applied load so that a point of unstable equilibrium at least one mode has a zero frequency.

For cases in which the mode of failure is collapse, it appears that a static analysis may be accepted as conservative in some situations. If the load is applied during a very short time interval, the inertial inhibits development of displacements, corresponding to those obtained from a static analysis under the same load. A static analysis is then generally conservative.

On the other hand if the load is rapidly applied and then sustained for a time that is at least of the same order of size as the period of the lowest vibration frequency, the static analysis is unconservative (it does not include the “overshoot”). In such a case it appears that static analysis based on twice the maximum load would be clearly conservative.

To answer the question of the need for a transient analysis, it appears that it often is useful and sometimes necessary to determine a few of the lowest vibration frequencies. If collapse and plastic deformation are excluded, these frequencies and corresponding “virtual masses” may be used directly in a modal transient analysis (not included in STAGS). Evaluation of the results of such an analysis may reveal whether or not a nonlinear transient analysis of the discrete system is needed (see discussion below).

**Is a linear analysis sufficient?**

In general, the results from a linear analysis are accurate if, under the design load, the squares of the rotations are small in comparison to the strains. However, large rotations, if confined to a narrow boundary layer, would have little effect on the shell behavior in general and do not preclude the use of linear analysis. On the other hand, the possibility of loss of structural stability must be considered even if the linear analysis results in small rotations. A bifurcation buckling analysis as discussed below would indicate a load level at which a rapid growth of the displacements may be expected.

Unless experience with similar cases is available to the user, it is suggested that the nonlinear static analysis be used as a matter of course. If the nonlinear effects are negligible, the design load an be applied and convergence will occur within a few iterations. In that case, the computer run time will exceed the time for a linear analysis only by some 10 to 30 percent because STAGS is based on the modified Newton–Raphson method. If convergence is not obtained, the linear solution is printed but a warning is served that it may be inaccurate. Usually, this indicates that the nonlinear terms at the applied load level are too large. An inspection of the results of the linear analysis may indicate that a nonlinear analysis is desirable.

Convergence failure may be caused by an ill–conditioned matrix. With a system that is almost singular, a solution will be obtained from the linear analysis that deviates from the true solution even if nonlinear terms are negligible. Such a situation may be revealed by inspection of the size of printed “equilibrium forces.” If there is doubt about the reason for nonconvergence, it may be advisable to rerun with the parameter \textit{ILIN} = 1 (M-1). Convergence failure in that case clearly indicates ill–conditioning and suggests that the model be reconsidered as discussed below. Whenever convergence fails and the equilibrium forces indicate that the equation system is satisfied the nonlinearities may be of significance.
In some cases, consecutive iterations tend to oscillate and nonconvergence at the design load may occur even if the linear analysis predicts stresses and displacements within a few percent. It appears that if the squares of the rotations are small in comparison to the strains and in addition a bifurcation buckling analysis indicates a large safety margin, then the linear solution is probably of sufficient accuracy even if nonlinearities cause nonconvergence. If the squares of the rotations are significant or if the bifurcation buckling analysis shows only a moderate margin of strength, a nonlinear analysis must be recommended.

In the case of transient response analysis by use of the explicit scheme the inclusion of nonlinear terms add little to the computer time.

*Can the stability problem be solved by use of a bifurcation buckling analysis?*

Loss of structural stability at a bifurcation in the load displacement path is discussed elsewhere. The bifurcation buckling theory gives information about the load level at which bifurcation occurs. In some cases loads can be applied far above the bifurcation buckling load without significant damage to the structure. In other cases the structure collapses well below this load due to imperfection sensitivity. Often experience (from experiments or from nonlinear analysis) indicates that the bifurcation buckling load is a significant design parameter.

Stability loss at a bifurcation point occurs only if the corresponding deformation mode is not contained in the deformation mode for arbitrarily small loads. For example, a flat plate with in–plane loading experiences no lateral displacements in the pre–buckling range. Thus it does not contain the lateral displacement modes of the buckled plate. Likewise, if the structure as well as loading is symmetric about some plane, all deformation modes antisymmetric with respect to that plane are possible bifurcation buckling modes. In such cases, we may choose to perform a buckling analysis with a nonlinear basic stress state. This question is further discussed under STRATEGY below.

Sometimes when a bifurcation point does not exist at all, the bifurcation buckling approach may still lead to an acceptable estimate of the critical load. A detailed discussion of this problem was presented in a 1972 paper by Almroth and Brogan. Briefly, it may be stated that the bifurcation approach is questionable (too conservative) if the structure is statically indeterminate and thus allows favorable redistribution of the stresses (e.g. shells with cutouts). The bifurcation approach gives unconservative results if the stiffness of the shell deteriorates with increasing load (e.g. long cylinders under bending).

The bifurcation buckling analysis with a linear stress state is probably a good approximation for any case in which the squares of the rotations in the linear solution are small in comparison to the membrane strains at the load level corresponding to bifurcation.

**Linear static analysis**

A linear stress analysis is straightforward and does not require any decisions about the computational strategy. If an adequate model of the structure has been defined and the grid is fine enough, a satisfactory solution is obtained unless the coefficient matrix is seriously ill-conditioned. This problem is discussed below.

**Eigenvalue analysis**

The strategy in the eigenvalue analysis is determined by the choice of suitable values for the parameters read on the D-2 and D-3 records. If vibration frequencies or bifurcation buckling loads are based on a nonlinear stress state, there are other decisions to make in connection with computation of the stress state. The strategy in such computations does not differ from the strategy in the regular nonlinear stress analysis discussed in a later paragraph.

In the case of nonlinear stress state, an option is provided to perform the stress analysis first, save the data for a certain number of load steps on tape and later decide for which of those load steps vibration frequencies or buckling loads should be obtained. This option may save some computer time. First, it may be easier to decide on the load levels at which eigenvalues are desired after the results of the stress analysis have been studied. The arrangement also makes it possible to find additional eigenvalues in a subsequent run. When eigenvalues are computed in a later run, the input data file for the nonlinear prestress analysis can be used with the exception that **IXEV** is set equal to 3 on the D-2 record. In addition, the user has the option of changing **NLDS** (D-2) and **PLDS** on the D-3 record to select certain of the data sets saved on tape for eigenvalue solution. Changes in boundary conditions are also permitted at this time.

For free vibrations in the presence of a nonzero stress state determined from a linear analysis, the same procedure can be used with **INDIC** = 5 on the B-1 record and **ILIN** = 1 on the M-1 record.

In a free vibration analysis, the eigenvalues correspond to vibration frequencies in [cps]. All eigenvalues are positive, except that those corresponding to rigid body motions, if such are allowed correspond to zero frequencies. The user can ask for a number of clusters of eigenvalues, each within a specified frequency range. In most practical applications the interest is confined to one range of eigenvalues, especially to a sequence of the lowest eigenvalues.
The user specifies the range of eigenvalues as well as maximum number of eigenvalues to be computed. If the I lowest eigenvalues are needed, the user can set EIGA=EIGB, SHIFT=0, and NEIG=I. If EIGA=EIGB and SHIFT is nonzero, then the NEIG eigenvalues closest to SHIFT will be determined. If EIGA < EIGB the analysis will be terminated when all eigenvalues in that range have been determined even if the number is less than NEIG.

If there exists a symmetry plane, in loading as well as in geometry, the size of the problem can be reduced and significant savings in the total computational effort can be achieved. If the structure on one side of the symmetry plane is considered, only the frequencies of symmetric modes are obtained. In a separate run with IBOND=1 on the P-1 record the antisymmetric modes can be determined. If the eigenvalue analysis is based on a nonzero prestress analysis with symmetry conditions and an eigenvalue analysis with boundary conditions corresponding to antisymmetry.

The eigenvalue approach for bifurcation buckling analysis with linear stress state is discussed elsewhere. This problem is slightly more complicated than the vibration problem because eigenvalues can be negative as well as positive. The NEIG eigenvalues closest to the point of shift (SHIFT or (EIGA + EIGB)/2) are found, independently of the sign.

Often the user is only interested in one eigenvalue—the lowest positive one. If the analysis is performed without a shift, it may happen that only negative eigenvalues are found because these are smaller in magnitude. In that case, the analysis has to be repeated with a positive shift. In choosing the shift for a repeated run the user can take advantage of the knowledge that the smallest positive eigenvalue is larger in magnitude than the largest of the negative eigenvalues that were found. Sometimes the buckling loads are symmetric with respect to zero. This is the case, for example, if a plate or a cylinder is subjected to uniform a shear load.

It may often be advisable to request more than one eigenvalue also in bucking analysis. If the structure shows insufficient strength and only the lowest eigenvalue and corresponding mode are known, reinforcements may be introduced that have little effect on secondary buckling mode with the eigenvalue below the design load.

It is usually possible to save time as well as computer cost by preliminary determination of approximate values for the buckling load under negative as well as positive load, before a large scale analysis is carried out. A linear analysis with a rather coarse grid will give some idea about the stress distribution. Once such a distribution is available it is usually possible to find “ball park” values by use of well known solutions for buckling of plates, spheres, or cylinders. Such knowledge is useful when the grid is defined, as well as in decisions regarding the shift.
The bifurcation buckling with a nonlinear prestress is of limited usefulness. Bifurcation buckling occurs only if the deformation mode corresponding to collapse is orthogonal to the deformation pattern in the basic stress state. If bifurcation takes place (into a mode that is permitted by boundary conditions), this will be apparent to the output from the nonlinear analysis since the coefficient matrix will have a negative root for each bifurcation point below that applied load. Introduction of geometric imperfections in the nonlinear analysis will remove the bifurcation an collapse will be indicated by a rapid growth of the collapse mode just before the critical load is reached. This problem is discussed under nonlinear analysis below. If at some level the nonlinear analysis is terminated due to the presence of a negative root, a bifurcation buckling analysis based on the stress distribution corresponding to a load factor somewhat below the critical load may be executed. This can be done, if in the nonlinear analysis some results have been saved—i.e., if \texttt{INDIC}=4 or 6 (B-1) and \texttt{IXEV}=0 (D-2). The results of such an analysis would reveal the approximate buckling mode. It is feasible that the negative root found in the nonlinear analysis is due to inaccuracy (ill-conditioning) in the computations rather than to impending buckling. This is probably the case if the computed eigenvalue far exceeds the value at which the determinant changes sign.

If the nonlinear analysis is terminated at the maximum load level, it is possible to gain some idea about the size of the margin of safety against collapse through the execution of an analysis of bifurcation from the nonlinear stress state. In that case the final solution is saved for subsequent bifurcation buckling analysis.

Another situation in which a bifurcation buckling analysis with nonlinear prestress may be useful arises when a symmetry plane exists. The situation in that case is the same as in the case of the free vibration analysis. The symmetry plane makes it possible to reduce the size of the problem only if buckling into antisymmetric modes is considered separately in a bifurcation buckling analysis with nonlinear symmetric prestress. It should be noticed that the computed eigenvalue gives a good estimate of the critical load only if the computed bifurcation load is close to the load level corresponding to the basic stress state. It may often prove practical to save data from a nonlinear run and attempt an antisymmetric bifurcation analysis somewhat below the collapse load for symmetric deformations. Such an analysis is likely to reveal whether the question of antisymmetric buckling need be further pursued.

**Nonlinear static analysis**

In the nonlinear stress analysis, a significant part of the computer time is devoted to the solution of a system of nonlinear algebraic equations. The computer time involved in those computations depends to some degree on the decisions the program user make regarding certain parameters on the D-1 record that governs the computational strategy in \texttt{STAGS}. 

Some understanding of the solution of those constants. Such understanding will also help the user in proper interpretation of the results and in avoiding certain pitfalls.

The solution of the nonlinear algebraic equations is usually based on the modified Newton–Raphson method. In general, solutions to the equations are obtained at a number of stepwise increasing load levels. The coefficient matrix is not necessarily reformed and factored when the load is increased, but attempts are made to obtain convergence by use of the factored matrix that is already available from computations at some earlier load step. The accuracy of the solution does not depend on the accuracy of the factored matrix. Two convergence criteria have to be satisfied before the solution is accepted. One of these requires that if the largest correction of a displacement unknown during the iteration divided by the largest displacement is less than a prescribed valued $\text{DELX} \times (D-1)$. The other criterion is applied to normalized values of the unbalanced forces, i.e., the vector representing the error in the individual equations.

The procedure converges only if reasonably good initial estimates of the unknowns are available. For the first load step, a linear solution is obtained and used as initial estimate. In the second load step, estimates are obtained by use of a linear extrapolation (based on the zero load solution and the first nonzero solution). From then on, estimates are obtained by use of a quadratic extrapolation. The user can suppress the extrapolation by use of the parameter $\text{NSTRAT} \times (D-1)$. Such action may, for example, be warranted if the analysis of the accuracy requirement is loosened by modification of $\text{DELX} \times (D-1)$. In order that convergence be obtained, neither the initial load factor nor the load step must be too large. Generally, the computer economy is best if the step is chosen so that convergence is obtained after two or three iterations. The first two load steps have less accurate estimates, and convergence is often somewhat slower. The number of iterations allowed in the effort to obtain convergence is internal to the code; it is not a user's option. In general, seven iterations are allowed. However, for the first load step, 15 iterations are allowed, unless the rate of decrease in consecutive corrections is so slow that continuation is meaningless. If convergence is not obtained on the first load step, the computations are discontinued so the user can reconsider his choice of initial load and maybe re-examine the discretization of the structure.

If nonconvergence occurs at a later stage in the analysis, the load step will be cut by a factor of 2, or the matrix will be refactored, depending on the values of the input parameters $\text{NCUT}$, $\text{NEWT} \times (D-1)$. The choice of these parameters is discussed below.

$\text{STAGS}$ assumes that nonconvergence has occurred if convergence is not achieved after seven iterations, if the solution is diverging for two successive iterations, or if the rate of convergence is indicated by two successive iterations is so slow that it is unlikely that the criterion will be satisfied within 7 iterations. If convergence has been obtained in the seventh iteration, then the following load step will be cut in half or a refactoring made
(depending on the user-selected strategy parameters \textbf{NCUT} and \textbf{NEWT}) before the iterations are started on that load step. Finally, if convergence is extremely easy, \textit{i.e.} if convergence occurs in one iteration on the present as well as the preceding load level, the step will be automatically increased through multiplication by a factor of 2.0.

The choice of the initial load level and the load step is the first decision the user has to make regarding the computational strategy. If it is expected that nonlinearities are of little influence at the design load, it may be possible to obtain convergence directly at that load level. In that case, the maximum load is set equal to the initial load and the step size is irrelevant. However, if a value of the collapse load is to be established or if there are considerable effects of nonlinearity at the design load, then the equations must be solved under a step-wise increasing load. In some cases, the nonlinearities are relatively insignificant until a point is reached that is close to the collapse load. In that case, a relatively large initial load and a smaller load step may be used. In other cases, such as cylinders with nonuniform external pressure or long cylinders under bending, convergence has been found to be difficult for relatively small loads. The initial load step may have to be as small as 5\% of the collapse load, and sometimes even less. If there is no special reason to proceed differently, it is suggested that the initial load as well as the load step is set equal to about one tenth of the estimated collapse load.

Very slow convergence with the modified Newton method is usually connected with oscillatory solutions. In that case it is recommended that a true Newton method be used, \textbf{NEWT} = -20 (D-1).

In most cases, the nonlinear analysis will not be completed in one run. Intermediate solutions are saved on tape or file so that the analysis can be restarted after the user has studied the results and reconsidered his strategy. The last three displacement solutions (if available) are saved on the tape so that a restart can begin with a quadratic extrapolation for initial estimates. Sometimes, as will be seen below, there may be reasons to restart from record one or two. At restart, the initial load factor must be the load factor corresponding to the record from which restart takes place. Before restart, it is advisable to ascertain that the data were properly recorded.

The input record with strategy parameters also includes a parameter \textbf{NSEC}. Occasionally, during computations, a check is made of whether the elapsed computer time exceeds \textbf{NSEC}; if so, intermediate results are saved on a data tape. \textbf{NSEC} should be chosen to be a few seconds less than the time estimate at which the operator aborts the run. Before each
refactoring, the program also checks that sufficient time is available to make refactoring meaningful. To refactor at the end of a run would be wasteful because a restart run always begins with factoring.

After initial convergence has been obtained, the program continues with the input load step until a load level is reached at which the analysis fails to converge within prescribed limits (or convergence is easy enough to allow load step increase). The program then either cuts the load increment in half or updates the factored matrix by use of the displacement vector solution obtained for the previous load step. The choice between these two alternatives is based on the following strategy parameters, input by the user:

- \textbf{NCUT} — Total number of times load increments may be cut in half.
- \textbf{NEWT} — Number of times the factored matrix may be recomputed.

On nonconvergence the normal action is reformulation and refactoring of the matrix. If convergence still fails the load step will be cut by a factor of two unless the step already has been cut \textbf{NCUT} times, in which case the program stops with the message that “convergence difficulties cannot be resolved.” If nonconvergence should occur after \textbf{NEWT} refactorings already have been performed, the load step will be cut until the number of load step cuts equals \textbf{NCUT}.

In view of the highly unpredictable nature of nonlinear behavior, it is very difficult to prescribe the best value for the strategy parameters in advance. As an initial choice for an unfamiliar type of shell, the values

\begin{align*}
\textbf{ICUT} &= 2 \\
\textbf{INEWT} &= 4
\end{align*}

are suggested. By observing the convergence behavior in previous computer runs, a more effective set of these values and the load step may be selected for continuation. In general, it seems advisable to restrict each computer run to some 10 to 20 times the computer time for one factoring so that the convergence behavior can guide the selection of strategy. In such a case, there should be little reason to control the number of factorings by \textbf{INEWT}. However, for very large systems with expensive factoring, especially in transient analysis, it may be better to continue an analysis with smaller steps than to repeatedly refactor the matrix. In that case \textbf{INEWT} may be set to 0, 1, or 2. \textbf{ICUT} essentially serves to define a minimum step size below which the user finds it impractical to perform the analysis. Choose \textbf{ICUT} such that the analysis will remain economically feasible.

The user may force factoring independently of any convergence difficulties. If \textbf{NEWT}=-N, then the matrix will be refactored at every Nth load step. Such a strategy is probably useful.
for problems with a relatively inexpensive. If convergence is slow, an over– or under–relaxation factor is automatically applied, depending on whether the convergence is uniform or oscillating. The user generally leaves blank fields for convergence parameters and relaxation factor. This means that the convergence parameter is applied as indicated above. The user may loosen the convergence criterion if it is indicated from the results that round-off errors make convergence difficult or even impossible. The criterion also may be tightened as subsequently discussed. The relaxation factor should be determined by the user only if he has considerable experience with the program. If the relaxation factor is given any value on the input record (including 1.0), this factor will be used throughout the analysis.

A collapse load is represented by a maximum in the load–displacement path. If a displacement is controlled and the load computed, it will be possible to compute points through and beyond the maximum. If, on the other hand, the load is the independent variable, instability is indicated by a failure to converge even with a very small load step. This, of course, is less satisfactory because convergence failure could have been caused by numerical difficulties rather than by rapidly growing displacements. Therefore, if convergence difficulties indicate that the collapse load is approached, the applied load may be plotted versus some representative displacement, the component that shows the greatest increase between the last two load steps may be selected. A better indication of collapse can be obtained if the load factor is plotted as a function of the determinant. This is possible only if a reasonably large number of refactorings have been executed. The value of the determinant is zero at the point of collapse for “dead load” but not for displacement or thermal loading. The best way to determine whether a limit point has been reached may be to restart the analysis in the dynamic mode. At the load level corresponding to nonconvergence or slightly higher the analysis is restarted as a transient analysis. In the presence of a limit point the solution vector will undergo large changes after only a few time steps.

The problems associated with the use of the load factor as independent parameter are related to solution prediction beyond the limit point and ill conditioning in the neighborhood of a limit point. Both these problems can be circumvented if some “path length” parameter is used as independent parameter. Such procedures were first published by Riks and are here referred to as the Riks method. While other procedures have been tried with some success, the Riks method is the most straightforward and clearly the most promising choice for introduction in the STAGS computer program. The path parameter introduced is the length of a vector with two components: a load parameter and some norm of the displacements. Since the components have physically different dimensions, they must be balanced in an appropriate way.

Some trial runs indicate that good results are obtained if the base load is chosen such that collapse will occur at a load factor of the order of unity. A procedure for selection of load
step size based on the “curvature of the path” seems to be working well. Only limited experience with the Riks method is presently available. However, in addition to the fact that computations can be carried beyond the limit point, it appears that this procedure is more efficient in terms of computer time.

For a dead weight (or a pressure applied by a compressible medium such as air) the true behavior of the structure at the limit point will be represented only by the results of a dynamic analysis. The limit point is first closely approached in a static analysis. Results from this analysis are used in a restart with increasing load and with dynamic effects included.

If a pressure is applied over a part of the structure through an incompressible medium such as water, the stability is not lost at the limit point and points on the falling part of the curve will represent meaningful results. However, these are difficult to determine. One method that sometimes can be used is to add fictitious springs so that the slope of the load displacement curve remains positive. The load carried by the springs is eventually subtracted out.

The existence of a bifurcation point along the primary path would be indicated by the vanishing of the determinant. Above the bifurcation point, the equilibrium on the primary path is unstable and the determinant is not positive definite: an attempt to refactor will result in a message that the matrix has one or more negative roots. This will occur only if the boundary condition in the model allows the corresponding bifurcation buckling mode.

Theoretically, as the critical load is reached, the round–off errors should trigger the deformation corresponding to the buckling mode. In practical application, it is generally found that in the early stage of buckling the amplitude of the buckling mode is so small in comparison to the displacement corresponding to the basic stress state that its growth, although in a relative sense large, will not violate the specified convergence criterion. This difficulty is avoided by the specification of initial imperfections in the shell geometry which are small enough not to appreciably affect the buckling load but large enough to trigger the new deformation pattern. The program, therefore, includes an option to add an initial lateral displacement pattern. This procedure has to be used, for example, if a study is made of post-buckling behavior of a shell of revolution subject to symmetric loading.

It has been found during use of the program that the imperfections may be useful as triggers also in other cases than those with perfect axial symmetry. For example, in the analysis of an elliptical cylinder with an aspect ratio of 1.5, it was found that without a trigger it was necessary to use a very severe convergence criterion $\varepsilon = 10^{-5}$ but if a small imperfection is added, the same collapse load may be computed in about half the run time with a less severe convergence criterion $\varepsilon = 10^{-3}$. If the elliptical cylinder has a significantly smaller aspect
ratio, it is likely that the amplitude of the buckling pattern which is present in the pre-critical displacement pattern is too small to act effectively as a trigger. In this case, the computation of a collapse load must include the use of a small imperfection. The choice of imperfection mode may often be aided by knowledge of the bifurcation bucking mode.

If difficulties like these do occur, they will be discovered when refactoring at a load level above the collapse load leads to a coefficient matrix which is not positive definite. In such a case, the user must either sharpen his convergence criterion or introduce an imperfection. If the imperfection is too small the unstable branch of the load-displacement curve for the perfect cylinder represents a configuration for which the equilibrium is almost satisfied. Since the convergence criterion cannot be made arbitrarily fine, this will at times lead to a false indication of convergence.

The run which ends with a message that negative roots are present, may sometimes be saved if a new run is restarted from an earlier solution. There also may be other reasons to suspect that an inaccurate solution has been accepted in which case restart from an earlier solution is advisable.

Through use of additional analyses with various degree of imperfections, the user of the program can get some notion of the imperfection sensitivity of the collapse load.

Finally, it may be advisable, often to set INDIC = 4 or 5 even if bifurcation or vibration analyses are not part of the original plans. Such a choice leaves the option open if the results of the nonlinear analysis indicate that a subsequent eigenvalue analysis may be helpful.

**Transient analysis**

A nonlinear transient analysis must be used if a structure that would be susceptible to buckling under static loading is subject to rapidly varying loads. However, such an analysis is generally quite expensive, and the user faced with the necessity may be willing to sacrifice accuracy in order to save on computer cost. It is necessary that the user is able to obtain some estimate on the computer run time before a decision is made on the procedure to be followed.

The strategy in the transient analysis is determined by a number of parameters defined on the E-1 and E-2 records. After defining a structural model and a preliminary grid the user must decide on the integration procedure.

Shell structures usually result in the so-called “stiff” system of differential equations— that is, the eigenvalues corresponding to free vibrations cover a very large range. The highest eigenvalues determine the numerical stability of explicit integration method. Therefore, a
very small time step must be used. On the other hand, it is relatively easy to determine the maximum time step. In the linear case the critical time step is given by

\[ \Delta t = \frac{2}{\omega_{\text{max}}} \]  \hspace{1cm} (15.1)

where \( \omega_{\text{max}} \) is the highest vibration frequency of the system. Usually the highest frequency does not vary much with the applied load in the nonlinear case. It is suggested therefore that in a nonlinear transient analysis with the explicit method the time step is chosen to be about 80 percent of that given by this equation. However, for cases in which geometric nonlinearities are important, it is likely that the lower frequencies of the system are the more important. In that case the implicit methods are generally more efficient.

Based on a finite difference formulation (of second order accuracy) for a shell of isotropic properties the critical time step is found to be

\[ \Delta t = \text{MIN} \left[ \frac{1}{c^2} \left( \frac{c^2}{\Delta \alpha} \right)^2 + \frac{c^2}{\Delta \beta} \left( \frac{1}{\Delta \alpha} \right)^2 + \frac{c^2}{\Delta \beta} \right] \] \hspace{1cm} (15.2)

where \( c^2 = \frac{E(1-v^2)}{\rho} \), \( c_s^2 = \frac{G}{\rho} \), and \( \rho = \) mass density of the material. The quantities \( \Delta \alpha \) and \( \Delta \beta \) represent grid spacings in arclength in the two directions such that \( \Delta \beta \leq \Delta \alpha \).

For layered and composite materials an average value of \( E \) can be used. If a higher order finite element formulation is used, the critical time step may be considerably smaller. With elements for which \( \Delta \alpha \), \( \Delta \beta \) are considerably larger than \( h \)—say twice as much or more—\( \Delta t \) according to equation (15.2) is probably reasonably accurate unless a refined membrane element is used. For refined membrane elements the critical time step may have to be decreased by a factor of two or more.

Except for some special cases then, equation (15.2) gives only an indication about where to start in a trial and error estimate of the critical time step. If the critical time step is exceeded computed displacement will grow very rapidly and the problem is noticed after only a few time steps. Instead of reliance on a trial and error procedure the user may take advantage of the knowledge that the highest frequency of the system is equal to or less than the maximum frequency for an individual element. Thus it is possible to choose the critical time step from an eigenvalue analysis for the smallest of all elements in the structure.

Notice that \( c \) is the velocity of a sound in the material. If in the one-dimensional case \( \Delta t \) is chosen to be exactly \( c \Delta \alpha \) then a stress wave progresses through the material with the speed of sound. Only for this value of \( \Delta t \) is the solution accurate (barring discretization
errors). If $\Delta t$ exceeds the critical value a small error will be magnified with each time step. If $\Delta t$ is less than the critical value error does not accumulate and the solution obtained is reasonably accurate except in case the loading is so severe that the magnitude of the nonlinear terms changes significantly during one time step.

Therefore, the explicit method is somewhat easier to use than any of the implicit methods. The latter are used for the purpose of reducing computer runtime. In particular, implicit methods are more flexible as they allow the choice of a larger time step with the purpose of sacrificing accuracy for a reduction in computer cost. With the explicit method no large systems of equations need be solved and no large matrices need be stored. Therefore, it may be possible to obtain solutions with the explicit scheme for cases that are beyond the capability of the implicit scheme (core or disk storage).

Several implicit methods have been provided in STAGS for use in transient response analysis. These are the trapezoidal formula, the Gear 2nd and 3rd order formulas and a formula due to K. C. Park. For a purely linear transient analysis ($\text{LIN} = 1$ on the M-1 record for all branches), the stability and accuracy of the trapezoidal formula make it the natural choice. When nonlinear effects are included in some part of the structure, the trapezoidal formula may become numerically unstable while the 2nd order Gear formula and the Park formula remain stable. The 3rd order Gear formula is also stable for sufficiently large time steps. Damping of higher frequency modes is substantially greater with the second-order Gear method. Consequently, the Park method is recommended for general nonlinear transient analysis. Figure 15.3 shows the damping in percent on each period as a function of $\Delta t \times f$.

![Figure 15.3](image-url)

\[ f = \text{FREQUENCY (Hz)} \]
\[ \Delta t = \text{TIME SKIP} \]
of time steps per period. However, the special properties of the Gear methods may be useful to users whose knowledge of their behavior is relatively sophisticated.

Typical for the implicit integration schemes is that a large time step results in inaccuracy in the form of artificial damping. Consequently the modes corresponding to the higher frequencies are damped out quickly. In order to obtain reasonably accurate results for the motion in the highest frequency modes, we must choose a time step that is of the same order of size as for the explicit integration. Therefore, implicit methods are not suitable for analysis of stress wave propagation.

If the problem is to determine whether the shell would collapse under some history of loading, the following procedure is recommended:

- Determine a number of deformation modes and frequencies for the stress-free structure. The number may depend on the case, say five to twenty modes. By use of a shift it may be possible to reduce the number of eigenvalues and vectors determined.
- The half period for the “highest mode” which must be reasonably accurately represented in the analysis is determined.
- Decide how much damping of this mode is acceptable and determine the time step from Figure 15.3.
- Check critical time step for explicit integration. If this is more than one-third of that determined above, use the explicit scheme. Otherwise use the K. C. Park method.
- Make an estimate of the number of time steps that would be required to complete the analysis.
- Approximate values for the average computer time per time step are given above.
- The total computer time for the analysis can now be determined. It seems prudent to estimate that at least twice as much time may be required due to false starts and wasted runs.
- If the estimate shows that the analysis is not economically feasible, several options may be considered such as: static analysis only, linear analysis, a coarser grid, other changes in the model, or a larger time step.

In the nonlinear transient analysis, the time step must be adjusted so that the accuracy is satisfactory without undue expense, and also it must be small enough to admit convergence at each stop. If nonconvergence occurs the matrix is always refactored. Should convergence fail after refactoring, the time step will be cut once.
Very little experience is available on the option with a variable time step. This part of the program must therefore be considered to be experimental. However, if the option is used, an unsuitable choice of the initial time step would be automatically adjusted. Thus, the choice of $\Delta T$ (E-1 record) is less critical. On the other hand, $\text{SUP}$ (E-1) must be defined since the error that is permitted at each time step is proportional to this value (the largest displacement to be expected, or rather the largest value of the maximum displacement value for which we still expect to obtain reasonable accuracy).

### 15.3 Progressive Failure Solution Strategy*

The progressive failure solution strategy implemented in STAGS leverages on the overall nonlinear solution algorithms already existing in STAGS. Composite damage modeling for laminated composite structures is treated in a manner similar to plasticity for metallic structures. The key difference here is the material degradation model for the composite damage models considered involves discrete reductions in lamina properties at material points (ply discounting as opposed to a flow rule). As an alternative to ply discounting for material degradation, another progressive damage model based on the crack-density state-variable approach developed by Prof. F. K. Chang of Stanford University has been implemented. Discrete local changes in stiffness cause difficulties for the nonlinear solution procedure. Chang’s approach includes some damping terms to stabilize the solution. The following sections describe the overall modeling aspects, choice of solution strategy parameters, maintaining equilibrium, and recommended guidelines.

**Modeling**

Modeling for composite damage simulations follows the usual STAGS approach for element and node generation (shell units or element units). Only the STAGS E410 and E480 shell elements have been tested. The material modeling is done through the GCP (generic constitutive processor) feature within STAGS. Some elements do not currently support the GCP option for material modeling (e.g., the E210 beam) and are restricted to STAGS’ historical approach using I- and K- records. With STAGS’ implementation of GCP, the composite damage modeling is available for orthotropic elastic brittle materials via the

---

* This section is based on material given in Knight, Norman F., Jr., “Enhancement of STAGS Progressive Failure Capability,” MRJ Technology Solutions Final Report for MRJ Task 1410, October 1999, pp. 32–37
ORT_EL_BR_MATERIAL option and for Chang’s progressive damage model based on the crack density state variable via the ABAQUS_UMAT_MATERIAL option.

The NGCP parameter on the B-3 record for the S1 processor indicates whether GCP input will be used (NGCP=1) or will not be used (NGCP=0). GCP material data libraries may be used exclusively in the model or in combination with K-record material data.

Input for the GCP material types begins with the l-5a input record using an alphanumeric command verb instructing STAGS to read specific GCP material data.

The input data for the ORT_EL_BR_MATERIAL command verb are defined on the l-10a record. This record gives the linear elastic material properties, the strain and/or strength allowables, the degradation parameter, some other material values, and identifiers to select the failure model and degradation model to be used.

The input data for the ABAQUS_UMAT_MATERIAL command verb are defined on the l-11a record. This record gives the linear elastic material properties, selected strength allowables, and various damage and damping model parameters required by Chang’s progressive damage model. The laminate data (layer material number, number of layer integration points per layer, layer orientation, and layer thickness) for both material types are specified using the l-21a GCP shell fabrication record.

For the ORT_EL_BR_MATERIAL material type, the user has the option of specifying nonzero positive values for either the strain allowables or the strength allowables or both. If the strain allowables are nonzero and the strength allowables are zero, then the strain allowables are used with the elastic constants to calculate strength allowables (e.g., $X_T = E_{11} \varepsilon_{11}$). Likewise if the strain allowables are zero, then they are computed internally based on the strength allowables. All values must be given as either zero or nonzero, and all values must be positive. Nonzero values must be given for all strain allowables or all strength allowables. Users cannot specify a mixed set of nonzero strain and strength allowables.

For the ABAQUS_UMAT_MATERIAL material type, the user must also provide several ASCII files in a specific subdirectory named OUTDIR. A unit–cell plane strain analysis and a curing residual thermal strain analysis generate these files. These files are generated by the execution of the PDCOMP and RSDE codes provided by Prof. F. K. Chang. Only the STAGS E410 shell element has been validated for this damage model.

Choosing solution strategy parameters

STAGS provides a number of parameters that are associated with the solution strategy. Incorporating the progressive failure capability into STAGS resulted in the addition of
several new features. This new capability has only been tested with the traditional solution control procedures (load control and displacement control). Parameters that affect the nonlinear solution process include \texttt{NCUT}, \texttt{NEWT}, and \texttt{DELX} on the D-1 solution parameter record and \texttt{NSOL} on the ET-1 solution control record used by the STAGS’ s2 processor.

The nonlinear solver in STAGS is a Newton–Raphson procedure for solving a system of nonlinear algebraic equations. The process involves a left-hand-side tangent stiffness matrix (second variation), a right-hand-side force imbalance or residual vector (first variation), an iterative increment in the displacement vector, an updating step for the iterative change in the displacement solution, and the formation of an initial guess or starting vector for the next solution step. These four parameters (\texttt{NCUT}, \texttt{NEWT}, \texttt{DELX}, and \texttt{NSOL}) affect these steps.

The \texttt{NCUT} parameter defines the total number of times the load/displacement factor (step size) can be cut before the solution process is stopped by STAGS. Generally, this parameter is set to a value between 5 and 10. Frequent cuts of step size generally indicate difficulties in the solution process; and careful examination of the results generated up to that point may reveal a modeling problem, a collapse behavior or some other event unforeseen by the user at the start of the analysis. A value of 5 is recommended until some familiarity with the problem is obtained.

The \texttt{NEWT} parameter defines the number of refactorings of the tangent stiffness matrix allowed during the execution. This parameter is discussed under the D-1 record and only a brief description will be given here. For \texttt{NEWT}=-20, the tangent stiffness matrix is evaluated, formed, assembled, and factored on each and every iteration. For \texttt{NEWT} = -1, the tangent stiffness matrix is evaluated, formed, assembled, and factored at the beginning of each and every solution step and possibly during the iteration process for that solution step as required by the solution strategy. However, it is always recomputed at the beginning of each solution step. For negative values of \texttt{NEWT}, the number of times the tangent stiffness matrix can be refactored is unlimited. For positive values of \texttt{NEWT}, the tangent stiffness matrix is recomputed as required by the solution strategy until a total of \texttt{NEWT} refactorings have occurred. With continuing increases in computer processor speeds and the availability of faster equation solvers, the trend towards using a true Newton procedure (update the tangent stiffness matrix on each iteration) is becoming a reality.

The \texttt{DELX} parameter defines the error tolerance on the iterative solution used as a stopping criterion for each solution step. After each iterative solution update, the norm of the iterative change in the displacement vector and the norm of the force imbalance or residual vector are computed and compared with the value \texttt{DELX}. If either of those norms is smaller than \texttt{DELX}, then the solution for that step is said to have converged. The default value of \texttt{DELX} is $10^{-3}$; however, typical values are usually between $10^{-4}$ and $10^{-6}$. In some cases, the default value may be used to get a solution and then a restart step performed using a smaller value.
of \texttt{DELX} to verify that an equilibrium state has actually been achieved. Setting \texttt{DELX} to a value larger than the default value is not recommended in general, and if done, requires the analyst to carefully examine the solution.

The \texttt{NSOL} parameter on the ET-1 record has been used primarily for restarts in the past. For the progressive failure capability, a new option (\texttt{NSOL}=3) was added. This option specifies that the previously converged solution will be used as the initial guess for the starting vector during the first two iteration of the next solution step. This option was developed in an attempt to avoid excessive amounts of new damage being created when an extrapolated initial guess is used to start the next solution step.

One solution parameter that users do not have control over is the number of iterations permitted per solution step, \texttt{ITLIM}. This is an internal parameter and is set to 11. For a progressive failure analysis in \texttt{STAGS}, this limit is \textit{automatically} doubled. If the iterative process is converging very slowly, then the tangent stiffness matrix will be refactored (provided the number of refactorings is not greater than \texttt{NEWT}) and/or the step size will be cut (provided the number of cuts is not greater than \texttt{NCUT}). Users may enter “\texttt{x_40_}” on the A-1 title record for the \texttt{STAGS s2} processor in order to allow a maximum of 40 iterations. This doubling of \texttt{ITLIM} appears to be necessary in some cases as the damage progresses during each iteration and may not have reached a stable damage state within 11 iterations. Some cases needed 15 to 20 iterations before no additional failures were detected for that solution step. Convergence of the nonlinear solution procedure is affected by the continual local stiffness changes as damage is detected, material properties degraded, and stress redistribution occurs.

During the iteration process, several variables are output and can be monitored. The residual error norm \texttt{RNORM} and the displacement solution norm \texttt{DNORM0} indicate the relative performance of the algorithm in achieving convergence. The solution strategy is controlled primarily by \texttt{RNORM}. In a full Newton–Raphson approach, the value of \texttt{RNORM} may increase up to a value of unity before the load factor is reduced. Once it reaches unity, the load factor is reduced regardless of the number of iterations.

In order to monitor this process, a new output parameter \texttt{TOTFAL} was added to the single line of output printed after each iteration cycle. This parameter gives the total number of points where failure has been detected up through that iteration. Once that value stabilizes (stops changing), no additional failures occur for this solution step. Then convergence to within the error tolerance \texttt{DELX} is relatively fast. If that value continues to increase, then additional failures are occurring and possible overall failure of the structure may be imminent. \texttt{TOTFAL} represents the total number of failed layer points in the model. It has a maximum value equal to the product of the number of layers, the number of surface integration points per element, and the number of elements with linear orthotropic elastic brittle and/or \texttt{ABAQUS}
UMAT materials. During iteration for the current solution step, TOTFAL should never be smaller than the value obtained from the last converged solution step. Each layer of the laminate has a specified number of layer integration points (see the I-21b record). If any or all layer integration points detect failure, this failure counts as only one layer failure point. For example, the maximum number of layer failure points for a 16-ply laminate is 64 for the E410 element with 4 surface integration points and 144 for the E480 element with 9 surface integration points—regardless of the number of layer integration points per layer.

Maintaining equilibrium

Establishing and re-establishing equilibrium for a nonlinear problem is a continuous problem during any nonlinear simulation even without brittle failures at the lamina level. The Newton–Raphson solution procedure is employed by STAGS to establish equilibrium at each solution step. The nonlinear solution strategy can be load control, displacement control or an arc-length-parameter control. Under load or displacement control, the load or displacement is incriminated and fixed during the iteration, and a new solution is found. Under arc-length control, the load and displacement are both changing during the iteration. So far, only the load and displacement control features of STAGS have been tested with the new progressive failure capability.

The Newton–Raphson procedure involves essentially four main stages. First, an initial guess for the next solution step is needed. Currently two options are provided: use the previously converged nonlinear displacement solution as the initial guess for the next solution step; and use previously converged solutions to perform a quadratic extrapolation for the next solution. If the step size is large, then the extrapolated initial guess may result in excessive amounts of damage and the solution may not converge. The load step would be cut and the iteration process continued until NCUTS cuts have been made.

Second, the second variation of the energy functional is needed which gives the “tangent” stiffness matrix (i.e., “tangent” only if evaluated using a converged nonlinear solution). This matrix can be evaluated, assembled, and factored on each iteration (full Newton–Raphson is obtained by setting NEWT = -20) or perhaps only at the beginning of each solution step (obtained by setting NEWT = -1) or if the error measures begin to increase rather than decrease (modified Newton–Raphson). To evaluate the “tangent” stiffness matrix, the material properties, the stress state, and the displacements are needed. The material properties for each iteration are initially those values archived after obtaining convergence for the previous solution step (i.e., this will be called the reference material state for this solution step). When the iterative change in displacements is computed and the displacement solution is updated, new strains are computed and new stresses are computed based on the archived material data. These stresses are then used in a failure analysis; and if failures are detected, the material properties are degraded. The stresses are then recomputed using these
degraded properties. At this point in a given iteration, the displacement solution has been updated; and the stresses and material properties are consistent with this solution. These values are then used to evaluate the “tangent” stiffness matrix for the next iteration. Then when a new displacement solution is obtained for the next iteration, the new stresses are computed based on the archived material data (i.e., the reference material state for this solution step)—not the material state from the previous iteration. Once convergence has occurred, the degraded material properties from the final iteration of the current solution step are archived to the historical database and used as the reference set of material data for the next step. In the modified Newton–Raphson procedure, the “tangent” stiffness matrix is updated periodically (e.g., at the beginning of each solution step (NEWT = -1)).

Third, the first variation of the energy functional is needed which gives the internal force vector and residual or force-imbalance vector. This vector is computed during every iteration for both the full and modified Newton–Raphson procedures. The computational process to evaluate the first variation is similar to the process used to evaluate the “tangent” stiffness matrix.

Finally, the displacement solution is updated on each iteration. Each iteration starts from the reference material state (i.e., the archived material data from the previous converged solution). The solution increment obtained by solving a system of equations based on the second variation and a right-hand-side vector based on the first variation is added to the solution vector from the previous iteration. As such, equilibrium is established and maintained throughout the solution process. Once a solution step has converged, the values of the degraded material properties for each material point with damage are saved (or archived) in the historical database as the converged material data for the next solution step. These data provide the reference material state for the next step.
**Recommended guidelines**

The following values for the solution parameters are recommended (based on limited testing):

- **NEWT** = -1 unless geometric nonlinearities are also very strong (*i.e.*, collapse of a shell)
- **NCUT** = 5 since a restart can be performed if necessary
- **DELX** = 0.000001 to ensure convergence and equilibrium for each step; **DELX** = 0.0001 is typically used for problems involving plasticity
- **NSOL** = 3 to reduce the overshoot in the solution and generate excessive damage in the model that is not really present
- **BETA** should be a small number if degradation is applied recursively (say 0.1; make it negative for degrade only once) or 0.000001 if it is only degraded once. Values equal to 0.1 appear to work well in either case but will generate some stress values that are not close to zero.
- The iteration output should be monitored and if the number of failures has not stabilized within 7 iterations, then a restart with “X_40_” on the A-1 title record for the STAGS s2 processor should be performed.
- Monitor the iteration history for each step. Once the total number of failed ply points (**TOTFAL**) stops changing, convergence should be very fast. If this is not true, then examine the value of the determinant (**DETERM**) and the degree of freedom associated with the largest residual term (**DOF**). These values are printed for each iteration. If these values are changing, then the solution is most likely still having trouble finding an equilibrium solution. If the values are not changing, an inconsistency in the implementation may have been detected.
- Post-process the results by plotting the percent of failed plies per element should be done to watch the damage progression. These contour plots are generated by setting the **ITEM** parameter on the PL-3 record for **stapl** to a value of 8 and the parameter **ICOMP** to zero. A “patch quilt” plot or element assignment plot can be generated for this parameter by including the character string “**DISCONT**” anywhere on the PL-1 record for **stapl**. In this case, an element is assigned a color based on the value of this parameter; no contouring across element boundaries occurs.
16

Interpretation of Results

An understanding of the definitions and sign conventions used in STAGS is crucial to interpretation of secondary solution data. The analyst will find this section an indispensable reference for understanding the meaning of the strains, curvatures, stresses, forces, and moments arising from both membrane action and flexure. Before going further, a few words about coordinate systems. Some solution data, such as Von Mises stresses and eigenvalues, are invariant with respect to coordinate transformation; but in general, post-processing involves transformation of both vector and tensor quantities into coordinate systems which are most meaningful to the application at hand. For this reason, STAGS gives the user great flexibility in specifying the system(s) in which results are to be expressed. Therefore, it is essential that the various coordinate systems used in STAGS be understood by the analyst. It will be helpful to refer to Section 4.1 “Coordinate Systems” as necessary to avoid any ambiguity regarding coordinate systems appearing in this discussion.

16.1 Shell Results

Figure 16.1 shows a curved shell with stress resultants (membrane forces) and moment resultants (flexural forces) expressed in shell coordinates \((x', y', Z')\). Figure 16.2 gives conventions for interpreting membrane forces, and Figure 16.3 gives conventions for interpreting flexural forces. Each of these last two figures shows a section of a curved shell whose dimensions are \(dx \times dy\). The \((x, y, z)\) coordinate system attached to this differential section is an arbitrary rectangular system in the plane of the shell. The \(z\) direction is normal to the shell at the point at which results are computed. Since the conventions for interpreting resultants are independent of the coordinate system in which they are expressed, it is not necessary to specify this \((x, y, z)\) system further. In practice, resultants are usually expressed in either material coordinates \((\phi_1, \phi_2)\) or fabrication coordinates \((\tau, \gamma)\), although the user may select any rectangular system in which the \((1, 2)\) axes are in the plane of the shell and the 3 axis is normal to the shell.
Refer to Figure 6.6 on page 6-58 for nodal-displacement conventions.
In **STAGS**, curvature is defined as

$$
\begin{align*}
\kappa_{xx} &= -w_{,xx} \\
\kappa_{yy} &= -w_{,yy} \\
\kappa_{xy} &= -2w_{,xy}
\end{align*}
$$

(16.1)
$D$, shell flexural rigidity, and $C$, a constitutive matrix, are defined for convenience as

$$D = \frac{Eh^3}{12(1-v^2)}$$

(16.2)

$$C = \begin{bmatrix} 1 & v & 0 \\ v & 1 & 0 \\ 0 & 0 & \frac{1-v}{2} \end{bmatrix}$$

where $h$ is thickness, and $E$ and $v$ are elastic constants.

Moments are then defined as

$$\begin{bmatrix} M_{xx} \\ M_{yy} \\ M_{xy} \end{bmatrix} = D \begin{bmatrix} \kappa_{xx} \\ \kappa_{yy} \\ \kappa_{xy} \end{bmatrix}$$

(16.3)

$$= \frac{Eh^3}{12(1-v^2)} \begin{bmatrix} 1 & v & 0 \\ v & 1 & 0 \\ 0 & 0 & \frac{1-v}{2} \end{bmatrix} \begin{bmatrix} -w_{xx} \\ -w_{yy} \\ -2w_{xy} \end{bmatrix}$$

In some references, the “twist” term ($\kappa_{xy}$) is defined as $\kappa_{xy} = -w_{xy}$, omitting the factor of 2. When this is done, $C(3,3) = 1 - v$. STAGS consistently uses the definition $\kappa_{xy} = -2w_{xy}$.

Transverse shear is defined as

$$Q_x = M_{xx,x} + M_{yx,y}$$

$$= D(-w_{xxx} - w_{xxyy})$$

(16.4)

$$Q_y = M_{xy,x} + M_{yy,y}$$

$$= D(-w_{xxy} - w_{yyyy})$$

Strains are defined as
where the superscript 0 refers to reference-surface strains, and \( z \) is the through-thickness distance from the reference surface (see “Effects of Eccentricity” on page 16-6) to the strain sampling point, the location where strains are computed.

Stresses follow from strains according to the following, where \( C \) is from (16.2).

\[
\begin{bmatrix}
\sigma_{xx} \\
\sigma_{yy} \\
\tau_{xy}
\end{bmatrix} = \frac{E}{1 - v^2} \begin{bmatrix}
[ C ]
\end{bmatrix} \begin{bmatrix}
\varepsilon_{xx} \\
\varepsilon_{yy} \\
\gamma_{xy}
\end{bmatrix}
\]

Interpretation of stresses can be inferred from the definition of resultants. The following equations define the relationships between stresses and resultants.

**Membrane Forces**

- normal forces

\[
N_{xx} = \int_{h}^{h} \sigma_{xx} dz
\]

(16.7)

\[
N_{yy} = \int_{h}^{h} \sigma_{yy} dz
\]

- planar shear forces

\[
N_{xy} = N_{yx} = \int_{h}^{h} \tau_{xy} dz
\]

(16.8)
**Flexural Forces**

- transverse shear forces

\[ Q_x = \int_{-h/2}^{h/2} \tau_{xz} \, dz \]

(16.9)

\[ Q_y = \int_{-h/2}^{h/2} \tau_{yz} \, dz \]

- bending moments

\[ M_{xx} = \int_{-h/2}^{h/2} \sigma_{xx} \cdot z \, dz \]

(16.10)

\[ M_{yy} = \int_{-h/2}^{h/2} \sigma_{yy} \cdot z \, dz \]

- twisting moments

\[ M_{xy} = M_{yx} = \int_{-h/2}^{h/2} \tau_{xy} \cdot z \, dz \]

(16.11)

**Effects of Eccentricity**

**STAGS** permits flexibility in positioning the shell wall with respect to the *reference surface*, the surface on which the nodes lie. The amount by which the *middle surface*, the surface defined by the center of the shell wall, is offset from the reference surface is known as *eccentricity*, and the analyst must be aware of its effects upon computed values of moment resultants. As shown in Figure 6.2 on page 6-28, the eccentricity, \( e \), is defined as the *Z’* (shell unit) or *z’* (element unit) value of the middle surface.

In equations (16.7) – (16.11), the limits of through-thickness integration are determined by the wall thickness, \( h \), and the eccentricity, \( e \). For example, if the reference surface coincides with the middle surface, then the integrals are of the form

\[ \int_{-h/2}^{h/2} f(\sigma) \, d\sigma \quad \text{or} \quad \int_{-h/2}^{h/2} f(\sigma) \cdot z \, dz . \]
Or, if the reference surface coincides with the bottom surface, then
\[
\int_0^h f(\sigma) dz \quad \text{or} \quad \int_0^h f(\sigma) \cdot z \ dz .
\]

Inspection of equations (16.7) – (16.9) show that computed values for force resultants are independent of eccentricity, but equations (16.10) – (16.11) show that computed values for moment resultants are not. For the case shown in Figure 6.2, a contour plot of moments can be misleading! This is not a matter of computing the \textit{correct} moments, but rather a subtle point regarding the \textit{interpretation} of moments. Fortunately, other results are not affected, and generally it is stresses and strains, rather than resultants, that are of primary interest.

The analyst may wish to “transfer” reference-surface moment resultants to the middle surface. This merely requires adding the effect of the eccentric membrane force resultants. This operation can be derived by considering the familiar expression
\[
\mathbf{M}_o = \mathbf{r} \times \mathbf{F}
\]
where \(\mathbf{M}_o\) is a moment vector at point \(o\) due to the eccentric force vector, \(\mathbf{F}\), and \(\mathbf{r}\) is a position vector defining the eccentricity of \(\mathbf{F}\) relative to point \(o\). Here, \(\mathbf{r}\) is the position of the reference surface relative to the middle surface, \(\mathbf{r} = (0\ 0\ -e)\), where \(e\) is the eccentricity (refer to Figure 6.2). It follows that

\[
\begin{bmatrix}
M_{xx} \\
M_{yy} \\
M_{xy}
\end{bmatrix}_m = \begin{bmatrix}
M_{xx} \\
M_{yy} \\
M_{xy}
\end{bmatrix}_r - e \cdot \begin{bmatrix}
N_{xx} \\
N_{yy} \\
N_{xy}
\end{bmatrix} \quad (16.12)
\]

where the subscript \(m\) refers to the middle surface, and \(r\) to the reference surface.

One final point regarding eccentricity—its effect on loading. During model development, the analyst should bear in mind that an eccentric membrane force is equivalent to a force plus a moment applied to the middle surface.
16.2 Beam Results

Figure 16.4 on page 16-10 shows a beam with resultant forces. \( V_y \) and \( M_z \) are due to bending about the \( z \) axis; \( V_z \) and \( M_y \) are due to bending about the \( y \) axis; \( T \) is due to torsion, or twisting about the \( x \) axis (not shown); and \( N \) is due to axial deformation. It is helpful to establish some terminology before continuing. Some of the terms and symbols used here are described elsewhere in this manual, and some are unique to beam results and are not found elsewhere.

Each beam element in STAGS is categorized using the following three classes:

- **ring**: a shell stiffener occurring along a row \((X = \text{constant})\) of a shell unit. Refer to Figure 6.5 on page 6-52 and records O-1a—O-1b, starting on page 6-50.

- **stringer**: a shell stiffener occurring along a column \((Y = \text{constant})\) of a shell unit. Refer to Figure 6.5 on page 6-52 and records O-2a—O-2b, starting on page 6-54.

- **beam**: a beam element in an element unit. Refer to Figure 8.5 on page 8-33 and record T-2 on page 8-9. Beam elements may also be defined via user-written subroutine USRELT.

Compare Figure 16.4 with Figure 6.5 and Figure 8.5 to help understand the distinctions among the three beam categories. Also refer to Figure 5.5 on page 5-122 regarding cross-section definition, and Figure 14.5 on page 14-13 for establishing the beam \((x', y', z')\) element coordinate system.

The element reference axis is established by the vector \( r_{12} \), which points from node 1 to node 2 (Figure 14.5). The beam cross-section axis, parallel to the reference axis, intersects the cross-section plane at the origin of the \((y, z)\) cross-section coordinate system. Referring to Figure 16.4, it can be seen that the reference axis is established by \( x' \) in a beam, \( X' \) in a stringer, and \( Y' \) in a ring. The cross-section axis is established by the cross-section coordinate \( \xi \). The \( x \) axis is not shown, but it is always parallel to the reference axis.

The cross-section axis is positioned using eccentricities, \( ECY \) and \( ECZ \), which offset it from the reference axis. The cross-section is oriented by rotating it through an angle \( XS1 \), indicated by \( \xi \) in Figure 16.4. \( ECY \), \( ECZ \), and \( XS1 \) thus establish the position and orientation of the \((y, z)\) cross-section coordinate system. Figure 16.4 shows an additional coordinate system, \((y, z)\), which is coincident with the unrotated \((\xi = 0)\) cross-section coordinate system. The \( x \) axis, coincident with \( \xi \) and parallel to the reference axis, is not shown. For all three beam categories, a positive \( ECY \) is directed from the reference axis in the positive \( y \) direction, and
a positive $ECZ$ is directed from the reference axis in the positive $z$ direction. Negative $ECY$, $ECZ$ are directed in the negative $y,z$ directions, respectively. $XSI$ is a right-handed rotation about the cross-section axis ($x$ or $\bar{x}$). The rotation vector points in the positive $\bar{x}$ direction for a positive $XSI$.

Beam results are expressed in the $(y,z)$ unrotated cross-section coordinate system. Figure 16.4 gives conventions for interpreting beam resultant forces, which are related to axial and torsional-shear stresses by the relations

$$N_x = \int_A \sigma_{xx}(y,z) \, dydz \quad M_z = \int_A y \sigma_{xx}(y,z) \, dydz \quad M_y = \int_A z \sigma_{xx}(y,z) \, dydz$$

$$T = \int_A \left[ y \tau_{xz}(y,z) - z \tau_{xy}(y,z) \right] \, dydz$$

Reference-axis strain, curvature, and unit twist are defined by the relations

$$\varepsilon_x = u_x \quad \kappa_z = -v_{xx} \quad \kappa_y = -w_{xx} \quad \alpha = (R_2^2 - R_1^2)/l$$

where $(u,v,w)$ are displacements in the $(x,y,z)$ directions, $l$ is the length of the beam, $R_1^2$ and $R_2^2$ are the axial rotations at nodes 1 and 2, and the unit twist ($\alpha$) has units of radians/length.

**Transformation of beam resultants**

Moments and curvatures may be transformed from the $(y,z)$ unrotated cross-section coordinate system to the $(\zeta, \eta)$ cross-section coordinate system according to

$$\begin{bmatrix} M_\zeta \\ M_\eta \end{bmatrix} = \begin{bmatrix} \cos \zeta & \sin \zeta \\ -\sin \zeta & \cos \zeta \end{bmatrix} \begin{bmatrix} M_z \\ M_y \end{bmatrix}$$

$$\begin{bmatrix} \kappa_\zeta \\ \kappa_\eta \end{bmatrix} = \begin{bmatrix} \cos \zeta & \sin \zeta \\ -\sin \zeta & \cos \zeta \end{bmatrix} \begin{bmatrix} \kappa_z \\ \kappa_y \end{bmatrix}$$

(16.13)
Interpretation of Results

Figure 16.4  Beam resultant forces. All forces are shown acting in their respective positive senses.

Conventions for Interpreting Beam Resultant Forces

- stress resultants, $N$, have units of force
- moment resultants, $M$, have units of $force \times length$
- flexural shear resultants, $V$, have units of force
- torsional shear resultants, $T$, have units of $force \times length$
- the right-hand rule applies consistently for interpreting moments
- positive normal force $N$ causes axial tension
- positive bending moment $M_z$ causes axial tension on the $+y$ surface
- positive bending moment $M_y$ causes axial tension on the $+z$ surface
- positive shear $V_y$ acts in the $+y$ direction on $+x$ faces, and in the $-y$ direction on $-x$ faces
- positive shear $V_z$ acts in the $+z$ direction on $+x$ faces, and in the $-z$ direction on $-x$ faces
- positive torsion, $T$, is directed in the $+x$ direction on $+x$ faces, and in the $-x$ direction on $-x$ faces
Axial stress and strain

Since the $x$ axis is coincident with the $z$ axis, there is no ambiguity regarding interpretation of axial stress and strain. Still, greater insight to the behavior of beams can be gained by considering the following relations among curvatures, moments, strains, and stresses.

Curvatures from (16.13) may be combined with axial strain at the cross-section axis ($\varepsilon^0_x$) to compute strain anywhere in the cross section according to

$$
\varepsilon_x = \varepsilon_x = \varepsilon_x^0 + \bar{y} \kappa_z + z \kappa_y
$$

(16.14)

Elastic stresses may then be computed as $\sigma_x = \sigma_x = E \cdot \varepsilon_x$.

Alternatively, when $(\bar{y}, z)$ are principal axes ($I_{yz} = 0$), elastic stresses may be computed according to

$$
\sigma_x = \frac{N}{A} + \frac{M_x \cdot \bar{y}}{I_z} + \frac{M_x \cdot z}{I_y}
$$

(16.15)

Torsional shear

STAGS assumes that torsional shear strain varies linearly with distance from the twist axis. In general, this assumption is valid only for circular cross sections. STAGS does not report torsional shear strain, but computes torsional shear stress using the familiar expression

$$
\tau = \frac{T_r}{J}
$$

where $r$ is the radial distance from the twist axis, and $\tau$ is the resultant shear stress acting in the circumferential direction.

Torsional shear results computed by STAGS should not be relied upon for non-circular cross sections. The torsional moment ($T$) and the unit twist ($\alpha$) are valid, however, and may be used to compute torsional shear stress by the analyst who has knowledge of stress distribution over a given cross section.

Flexural shear

STAGS computes flexural shear resultants, $V_y$ and $V_z$. No assumption is made regarding cross-sectional variation of shear strain, however average values $(\bar{\gamma}_y, \bar{\gamma}_z)$ are computed.
Flexural shear stresses are not reported. $V_y$ and $V_z$ may be used to compute flexural shear stress by the analyst who has knowledge of stress distribution over a given cross section.

**Combined shear**

Net shear strains $(\gamma_{xy}, \gamma_{xz})$ and stresses $(\tau_{xy}, \tau_{xz})$ may be computed by combining flexural and torsional results, where the analyst has computed these. STAGS does not compute cross-sectional variation of shear in beams, which is a complex matter for all but the simplest of situations, such as a circular cross-section under pure torsion or a rectangular cross-section under pure flexure.

If rigorous treatment of beams is important, they should be modeled using shells. For example, a Tee-stiffener can be modeled using separate shell branches for web and flange.

### 16.3 Evaluating Solution Quality

### 16.4 Interpreting Diagnostic Messages

### 16.5 Overcoming Difficulties

It is important that the output be carefully studied. For example, even if only the bifurcation buckling load is of direct interest, it is worthwhile to study carefully the buckling mode. Such a study may reveal a flaw in the modeling which makes the computed critical load questionable. In general, a study of the output may indicate whether a suitable grid has been chosen. It is a good habit to look through the entire output even if everything appears to be all right.

Arrangements have been made so that the program can be used as a preprocessor of data records. If the program module S2 is excluded, all input records are read, error checks are made, but no time-consuming computations will be carried out. The printout from such a run contains all of the messages that are based on the input data directly including possible error messages. It also contains the matrix of shell wall stiffnesses. A plot of the grid can be obtained in an isometric view. Because such a run generally executes in a few seconds and only requires minimum computer core storage, it can be made on a high priority basis as a preliminary to a longer low priority run.
Error messages are obtained when various indices are outside of their allowed or specified range. In many cases, input record errors will cause the remaining data records to be read in an incorrect sequence which will normally cause the run to abort. In such a case, additional errors will not be found unless the run is repeated after correction of the error.

Error messages may appear embedded in the regular output. If the run is aborted for reasons that are not immediately clear, look through the output for such messages.

The following text discusses some special problems that may occur and their possible solutions. Computer system problems are not included in this discussion.

**Linear static analysis**

**Problem** Output from matrix decomposition indicates that the stiffness matrix is singular or near-singular.

**Action** The stiffness matrix becomes singular when rigid body motion is not prevented. Errors or omissions in data defining wall construction and material properties may lead to a singular system. It may also be noted that as the mesh spacing is reduced, the stiffness matrix will approach a singular condition in the limit.

**Problem** The displacement or stress solution appears not to vary smoothly with the shell coordinates.

**Action** One possibility is that the grid is too coarse. It is also possible that the solution is inaccurate because of roundoff errors. Check equilibrium forces! An accurate solution possibly can be obtained by use of nonlinear analysis as a form of linear refinement. If convergence is not obtained in a nonlinear analysis, look for reasons for ill-conditioning and try to improve the model.

**Problem** The displacements appear to be too large.

**Action** An error in the model or in the shell stiffness is conceivable. It is also possible that the boundary conditions allow rigid body motions. Check equilibrium forces!

**Problem** The displacements are all zero.

**Action** Probably the case has been specified so that there is no load. Check if the load is in conflict with boundary conditions, *e.g.* applied at a point at which the boundary conditions do not allow displacement.
Problem  The average bandwidth of the stiffness matrix is zero and the determinant is exactly 1.0.

Action  These symptoms occur when there are no nonzero terms in the stiffness matrix. The problem is probably caused by errors in the data defining the wall properties. The stiffness matrix will also vanish identically when all freedoms of the system have been constrained by boundary conditions or specified displacements in the load data.

Problem  The stress at some point does not appear to converge with the grid size.

Action  A singularity may exist. If it is necessary to find accurate values for the stress at the point in question, a special analysis based on three-dimensional inelastic analysis may have to be carried out for a part of the structure in the immediate vicinity of the singularity. If displacements appear to converge, the displacement solution and the stresses away from the singularity are probably satisfactory.

Eigenvalue analysis

Problem  The output indicates the existence of negative roots when the matrix is refactored after the shift. (If negative roots occur without shift, see linear analysis above.)

Action  Each negative root represents an eigenvalue between zero and the value of the shift. If the number of negative roots is larger than the number of computed eigenvalues between zero and the value of the shift, there must be additional eigenvalues between zero and the smallest computed eigenvalue. A rerun is then needed if the lowest eigenvalues are required.

Problem  Only negative eigenvalues were obtained, though a positive value was desired.

Action  It is necessary to use a positive shift.

Problem  The buckling or vibration mode shows jumpiness in the displacement, changing sign at every grid point.

Action  The grid is probably too coarse and should be modified. The occurrence of such buckling modes is discussed under STRATEGY.
Problem  No convergence on all or some of the requested eigenvalues.

Action  The program iterates 30 times. This is usually sufficient. Should nonconvergence occur, the intermediate values of the eigenvalues (if printed) will give some information. They indicate approximate values of buckling loads or vibration frequencies. A repeated analysis with a shift closer to the nonconverged eigenvalues will probably remedy the problem. It may be useful to ask for more than one cluster of eigenvalues in the repeat run, so that none of the eigenvalues to be determined are too far away from a point of shift.

Nonlinear static analysis

In the nonlinear analysis, a normal exit can occur in several different ways. Normal exit does not always mean that the analysis has been successfully completed. The program output advertises when one of the following normal exits occurs:

- Convergence not obtained at first step.
- The time limit NSEC is exceeded or there is not enough time left to perform the next option.
- The maximum load is reached.
- Convergence not obtained and the values set on the parameter NEWT and NCUT have been reached.
- Convergence difficulties cannot be resolved if the maximum number of cuts is reached and nonconvergence occurs after refactoring.

An abnormal exit can occur for several reasons, none of which can be indicated by the program output. Examples of abnormal exits are:

- Maximum time on control record has been exceeded.
- Maximum I/O time allowed has been exceeded.
- Maximum allowed computer core storage or disc storage exceeded.
- Systems error.
- Error in user-written subroutine.

If the analysis is to be continued, check in the system output that the appropriate data have been saved on tape or disc file.
**Problem** No convergence on first load step for an original run (not restart).

**Action** The cause may be that the system is ill-conditioned or that the effect of nonlinearities is too large. Use of a less severe convergence criterion is possible if the ill-conditioning is moderate. However, this may lead to an unnecessarily expensive analysis as convergence difficulties will be recurring. If too much nonlinearity prevents convergence, this may be caused by a mistake in modeling or in the data input. Nonconvergence will usually result if rigid body motion is permitted, even though applied loads are in balance.

**Problem** No convergence on first load step of a restart.

**Action** This may be because a load level has been reached that is close to a limit point. If the load displacement curve based on the previous runs does not indicate clearly that collapse is about to occur, continue the analysis, with much smaller steps. The nonconvergence also may be caused by an error in input data. It should be determined also whether the initial load factor corresponds to the solution on the selected restart record and whether the previous run was properly saved; check `ISTART` and `STLD`!

**Problem** The output indicates that convergence difficulties cannot be resolved.

**Action** This may not be a problem. Such output is obtained when convergence fails if the number of cuts of load step has reached `NCUT`, but the number of refactorings is still below `NEWT`. It would not serve any purpose to do additional factorings without a cut in the load step. If it appears that the run should be continued, it must be restarted with a smaller load step.

**Problem** At refactoring the determinant is found not to be positive definite (this could happen at restart or during the run).

**Action** This is an indication that either there is a bifurcation point on the primary path below the load level is reached, or that an inaccurate solution was accepted at a previous step. If there is a true bifurcation, the analysis can be rerun with a small imperfection (or perhaps a small lateral load) which will trigger displacements in the critical mode. If a true bifurcation does not exist, there may be present a displacement mode that has an extremely small amplitude at moderate loads. At some load level (close to a sharp maximum in the load displacement curve) this amplitude begins to grow rapidly. In that case either an imperfection may be used or the convergence
criterion may be sharpened. This is discussed under strategy for nonlinear analysis.

**Problem** Convergence difficulties seem to persist even with moderate nonlinearity.

**Action** The system may be ill-conditioned; the accuracy of the computations are marginal in comparison to the convergence criterion. This problem may be cured by use of a less severe convergence criterion. However, with a coarser criterion, less accurate solutions will be accepted, and the predictions for initial estimate will be inaccurate. The output indicates for each iteration at which point the largest change is displacement occurs. If the convergence is nonuniform oscillating, it may be practical to switch to a true Newton method.

**Problem** At some load level, the displacement solution does not appear to be smooth.

**Action** A jumpy solution probably indicates that a spurious mode of collapse is developing. Use a finer grid!

**Transient analysis**

**Problem** Convergence difficulties or negative roots.

**Action** See Nonlinear Analysis.

**Problem** The explicit scheme is used and the displacement components grow extremely rapidly.

**Action** Mathematical instability occurs because the time step is too large. Rerun with smaller time step!

**Problem** One of the implicit integration methods is used. The motion of the structure seems to indicate unwarranted damping.

**Action** The time step is probably too large. Reduce the time step or use the option with a variable time step. If the problem occurs with a variable time step, reduce the value of $\text{SUP (E-1)}$.

**Problem** The third order Gear method is used. The motion of the structure seems to indicate that energy is added to the system in a way that is not due to the
work of external forces. A vibration with increasing amplitude in the lowest frequency mode is a reason for suspicion.

**Action**
The chosen time step is such that at least one of the deformation modes is in the unstable range. Choose other integration method, modify (increase) time step, or add some structural damping.
STAGS uses various file systems; e.g., text input/output files, binary database files, and scratch files. Understanding these file systems is important for effective use of STAGS. This is especially so for restarting an analysis, for pre/postprocessing, and for interfacing with the database via STAR, the STAGS Access Routines.

There are two sections in this chapter. Section 17.1 describes each file, and section 17.2 summarizes file requirements according to the various STAGS processors.

See Section 3.5 “File Systems” on page 3-8 for additional information on files.

17.1 Description of I/O Files

This section gives a brief description of each STAGS file. The files appear in alphabetical order.

ACCEL casename.accel.i

The ACCEL file, created by stp per user request, contains text data in PATRAN “results” format representing the nodal accelerations at load/timestep i. See $STAGSHOME/doc/stpfiles for more information.

BIN casename.bin

The BIN file contains input to the STAGS solution processor, s2. Solution input data are described in Chapter 11.

BSR casename.bsr

The BSR file is a binary scratch data file that is present only during execution of certain STAGS processors. If an orphan BSR file is found after
all processors have finished, it should be deleted. BSR files can grow quite large!

**DISP** \textit{casename.disp.i}

The \textit{DISP} file, created by \texttt{stp} per user request, contains text data in PATRAN “results” format representing the nodal displacements at load/time step \textit{i}. See \$\texttt{STAGSHOME/doc/stpfiles} for more information.

**EGV** \textit{casename.egv}

The \textit{EGV} file contains a binary representation of eigenvector data for selected load/time steps during each run or execution of the \texttt{STAGS} processors. The \textit{EGV} file is indexed by the \textit{IMP} file.

**EGV.bak** \textit{casename.egv.bak}

The \textit{EGV.bak} file contains a backup copy of the \textit{EGV} file. The \textit{EGV.bak} file is copied from an \textit{EGV} file when \texttt{s2} is run and an \textit{EGV} file for that case exists. Only one \textit{EGV.bak} file exists at any time. The user can suppress the creation of the \textit{EGV.bak} file by specifying the appropriate option with the \texttt{stags} command.

**EGV.BOMB** \textit{casename.egv.BOMB}

The \textit{EGV.BOMB} file contains the most recently obtained eigenvector data when the \texttt{s2} process terminates abnormally. Often the \textit{EGV.BOMB} file will be usable, but since the case terminated abnormally, there is no guarantee. Usually an \textit{EGV.BOMB} file will contain load/time step data up to the step that triggered the abnormal termination. In the event the user encounters an \textit{EGV.BOMB} file, he/she must decide whether to use it or the \textit{EGV.bak} file that is normally available before performing further analysis.

**EIG** \textit{casename.eig.i.j}

The \textit{EIG} file, created by \texttt{stp} per user request, contains text data in PATRAN “results” format representing a normalized displacement due to eigenmode \textit{j} for loadstep \textit{i}. See \$\texttt{STAGSHOME/doc/stpfiles} for more information.

**elmI** \textit{casename.elmI.i}

The \textit{elmI} file, created by \texttt{stp} per user request, contains text data in PATRAN “results” format representing stress and moment resultants for solution time/load step \textit{i}. See \$\texttt{STAGSHOME/doc/stpfiles} for more information.
**elmP**  
*casename.elmP.i.j*

The *elmP* file, created by *stp* per user request, contains text data in **PATRAN** “results” format representing plastic strains for solution time/load step *i* in material layer *j* for those cases where plastic strains exist. See $STAGSHOME/doc/stpfiles for more information.

**elmR**  
*casename.elmR.i*

The *elmR* file, created by *stp* per user request, contains text data in **PATRAN** “results” format representing reference-surface strains and curvatures for solution time/load step *i*. See $STAGSHOME/doc/stpfiles for more information.

**elmT**  
*casename.elmT.i.j*

The *elmT* file, created by *stp* per user request, contains text data in **PATRAN** “results” format representing strains and stresses for solution time/load step *i* in material layer *j*. See $STAGSHOME/doc/stpfiles for more information.

**FINT**  
*casename.fint.i*

The *FINT* file, created by *stp* per user request, contains text data in **PATRAN** “results” format representing the nodal forces at load/time step *i* for the given **STAGS** model. See $STAGSHOME/doc/stpfiles for more information.

**FORT**  
*fort.*

*TMP* and *FORT* files are miscellaneous scratch files and should disappear upon termination of the process that created them. If you are sure no **STAGS** processes are running in the current directory and these files exist, you may delete them.

**IMP**  
*casename.imp*

The *IMP* file contains an index of the latest set of data contained in the **EGV** file.

**IMP.bak**  
*casename.imp.bak*

The *IMP.bak* file contains a backup copy of the *IMP* file. The *IMP.bak* file is copied from an *IMP* file when *s2* is run and an *IMP* file exists for that case. Only one *IMP.bak* file exists at any time. The user can suppress the
creation of the IMP.bak file by specifying the appropriate option with the stags command.

**IMP.BOMB**  casename.imp.BOMB

The IMP.BOMB file contains the most recently obtained eigenvector data when the s2 process terminates abnormally. Often the IMP.BOMB file will be usable, but since the case terminated abnormally, there is no guarantee. Usually an IMP.BOMB file will contain load/time step data up to the step that triggered the abnormal termination. In the event the user encounters an IMP.BOMB file, he/she must decide whether to use it or the IMP.bak file that is normally available before performing further analysis.

**INP**  casename.inp

The INP file contains input to the STAGS model processor, s1. Model input data are described in Chapters 5–10.

**LOD**  casename.lod

The LOD file is a binary scratch file used by stp to store selected displacement data. It is deleted upon normal termination of stp.

**LOG**  casename.log

The LOG file contains text diagnostics from the latest stags command.

**MDL**  casename.mdl

The MDL file is a PATRAN Neutral File that is created by the STAGS-to-PATRAN model translation function of stp.

**OUT1**  casename.out1

The OUT1 file contains text output from the STAGS processor s1.

**OUT2**  casename.out2

The OUT2 file contains text output from the most recent invocation of the STAGS processor s2.

**OUT2.i**  casename.out2.i

An OUT2.i file results when the STAGS processor s2 is run through the stags command and an OUT2 file exists. The existing OUT2 file is
renamed to an OUT2.i file where \( i \) is either 1 or the greatest \( i \) plus 1 of existing OUT2.i files.

**Examples**

If the file ring.out2 exists and we run s2 through the \texttt{stags} command, ring.out2 will be renamed ring.out2.1 and the output from the latest execution of s2 will be placed in the file ring.out2.

If the following files exist: ring.out2, ring.out2.1, ring.out2.2, and ring.out2.3 and if s2 is executed through the \texttt{stags} command, the file ring.out2 will be renamed to ring.out2.4 and output from the latest execution of s2 will be placed in the file ring.out2.

If the files ring.out2.1 and ring.out2.2 exist but ring.out2 does not exist and if s2 is executed through the \texttt{stags} command, output will be placed in the file ring.out2, and no renaming will occur.

Once an OUT2 file becomes an OUT2.i file, no further renaming occurs.

The OUT2 file always contains the latest output of an s2 execution. Previous executions of s2 on the same case are placed in numerical order in OUT2.i files from oldest, \( i = 1 \), to second newest, \( i = j \), where \( j \) is the maximum of all \( i \)'s.

**Note:** OUT2 files are renamed to OUT2.i files in the fashion described above only if s2 is executed through the \texttt{stags} command or if the user explicitly renames the files.

**PDF**

\texttt{casename_m.pdf} and \texttt{casename.pdf}

These PDF files contain plots generated in Adobe’s Portable Document Format (pdf) by STAGS’ \texttt{stapl} plotting processor for the model and selected solutions, respectively.

**PIN**

\texttt{casename.pin}

The PIN file contains input for STAGS’ \texttt{stapl} plotting processor.

**PLOT**

\texttt{casename.i.plot}

The PLOT file is a text file created by the \( i^{th} \) invocation of the “xy” function from within a single \texttt{stp} session. The PLOT file may be used directly as input to xgraph or plotps.

The integer \( i \) is used to age the PLOT files. If no PLOT file exists then \( i = 1 \), otherwise \( i = j+1 \) where \( j \) is the highest-numbered existing PLOT file. See \texttt{$STAGSHOME/doc/stpfiles} for more information.
**Input/Output Files**

**Description of I/O Files**

**POUT**

`casename.pout`

The *POUT* file contains printed output from the *stapl* plotting processor.

**PS**

`casename_m.ps` and `casename.i.ps`

*PS* files contain data for plots generated in Adobe’s *PostScript format* (ps) by the *stapl* program; they can be printed directly on a PostScript printer or viewed in a PostScript viewer.

The `casename_m.ps` file is a *PostScript-formatted* plot—generated by STAGS’ *s1* (model definition) processor—of the undeformed model.

Each `casename.i.ps` file contains one plot of the undeformed or deformed configuration, with or without solution information, generated by the *stapl* processor; the integer *i* is used to age the *PS* files; if no *PS* file exists then *stapl* sets *i = 1*, otherwise it sets it to *i = j + 1*, where *j* is the highest-numbered existing *PS* file.

**PXY**

`casename.pxy`

The *PXY* text file contains y(x) data generated by STAGS’ *xyrans* processor; these files facilitate visual inspection of STAGS results and can often be used effectively as (part of the) input into an xy plotting program.

**RES**

`casename.res`

The *RES* file contains a binary representation of the solution state for the load/time steps evaluated in the current, and if applicable, previous executions of the STAGS processor *s2*. The *RES* file is indexed by the *RST* file. The *RES* file is created the first time *s2* is run for a given case. Each subsequent run of *s2* for that case appends and/or overwrites data to the *RES* file.

**RES.bak**

`casename.res.bak`

The *RES.bak* file contains a backup copy of the *RES* file. The *RES.bak* file is copied from an *RES* file when *s2* is run and an *RES* file for that case exists. Only one *RES.bak* file exists at any time. In case the *RES* file is very large (i.e. more than 1/2 available free disk space), the user can suppress the creation of the *RES.bak* file by specifying the appropriate option with the *stags* command.

**RES.i**

`casename.res.i`

When the user selects the “pcopy” function of *stp*, he or she is asked to select a subset of load/time steps from the current *RES* file and copy them...
into a new RES file that is usually much smaller. The original RES file is renamed to an RES.i file where i is either 1 or the highest plus one of existing RES.i files. The file RST.i indexes the database file RES.i.

**RES.BOMB casename.res.BOMB**

The RES.BOMB file contains the current database when the s2 process terminates abnormally. Often the RES.BOMB file will contain usable data; but since the case terminated abnormally, there is no way to guarantee that the database is intact. Usually a RES.BOMB file will contain load/time step data up to but not including the step that triggered the abnormal termination. In the event the user encounters a RES.BOMB file, he/she must decide whether to use it or the RES.bak file that is normally available before performing further analysis.

**CAUTION:** If a truly bad RES.BOMB file is renamed to a RES file and the creation of a RES.bak file is suppressed during an s2 execution, the database could be lost.

**RST casename.rst**

The RST file contains an index of the RES file for the load/time step solutions generated from running the STAGS processor s2.

**RST.bak casename.rst.bak**

The RST.bak file contains a backup copy of the RST file. The RST.bak file is copied from an RST file when s2 is run and an RST file for that case exists. Only one RST.bak file exists at any time. The user can suppress the creation of the RST.bak file by specifying the appropriate option with the stags command.

**RST.i casename.rst.i, where i is an integer**

When the STAGS user selects the “pcopy” function of stp, he or she is asked to select a subset of load/time steps from the current RST file and copy them into a new RST file that is usually much smaller. The original RST file is renamed to an RST.i file where i is either 1 or the highest plus one of existing RST.i files. The file RST.i indexes the database file RES.i.

**RST.BOMB casename.rst.BOMB**

The RST.BOMB file contains an index for the current database when the s2 process terminates abnormally. Often the RST.BOMB file will be usable, but since the case terminated abnormally, there is no guarantee. Usually an RST.BOMB file will contain load/time step data up to the step
that triggered the abnormal termination. In the event the user encounters an *RST.BOMB* file, he/she must decide whether to use it or the *RST.bak* file that is normally available before performing further analysis.

**SAV** 
*casename.sav*

The *SAV* file, a binary representation of the **STAGS** model, is created by the **STAGS** processor **s1**. One can also create a *SAV* file by making the appropriate calls to the “data put” (DP) routines in the **STAR** library.

**SOT** 
*casename.sot.i*

The *SOT* file contains text diagnostics from running a program that calls the **STAR** library. The file is created upon a call to DUOPEN and upon a normal call to DUCLOSE, the file is deleted.

**STEP** 
*casename.step*

The **STEP** file contains a complete listing of the load/time step history contained in the *RES* file. The **STEP** file is created by various postprocessors.

The integer *i* is used to age the *SOT* files. If no *SOT* file exists then \[ i = 1 \]
otherwise \[ i = j + 1 \] where *j* is the highest-numbered existing *SOT* file.

**TMP** 
*tmp.*

**TMP** and **FORT** files are miscellaneous scratch files and should disappear upon termination of the process that created them. If you are sure no **STAGS** processes are running in the current directory and these files exist, you may delete them.
17.2 Summary of I/O File Requirements

The following sections discuss input files required for each STAGS processor and the output files generated or modified. Files marked with an asterisk (*) are read and/or written under certain conditions only. See the file descriptions above for details. Files marked with a hash (#) are temporary files that may be present while the processor is running but which should disappear upon termination.

**s1—Model processor**

**Input:**

- INP
- EGV*
- IMP*

**Output:**

- LOG
- OUT1
- SAV
- BSR#
- TMP#
- PDF (casename_m.pdf) or PS (casename_m.ps)

**s2—Solution processor**

**Input:**

- BIN
- SAV
- EGV*
- IMP*
- RES*
- RST*

**Output:**

- LOG
- OUT2
- RES
- RST
- EGV*
- IMP*
- OUT2.i*
- BSR#
- FORT#
- TMP#

**stapl—Plotting processor**

**Input:**

- PIN
- SAV
- RES*
- RST*

**Output:**

- PDF
- PS
- POUT

**stp—STAGS translator/postprocessor**

**Input:**

- PCN*
- RES*
- RST*
- SAV*
- PATRAN Neutral File *(input to PAT2S, Appendix X)

**Output:**

- ACCEL*
- DISP*
- EIG*
- elmI*
- elmP*
- elmR*
- elmT*
- FINT*
- MDL*
- PCN*
- PLOT*
- PS*
- RES*
- RES.i*
- RST*
- RST.i*
- SAV*
- SOT*
- STP*
- VEL0*
- BSR#
- FORT#
- LOD#
- PD#
- TMP#
STAGS Input Record Catalog

Model Input—INP File

Summary and Control Parameters

A-1 Case Title
COMMENT

B-1 Analysis Type Definition
IGRAV ICHECK ILIST INCBC NRUNIT NROTS KDEV

B-1a Model Unit List (for plotting)
( IUNIS(i), i = 1, NRUNIT )

B-1b Sequence of Model Rotations (for plotting; NROTS records required)
IROT ROT

B-2 General Model Summary
NUNITS NUNITE NSTFS NINTS NPATS NCONST NIMPFS INERT NINSR NPATX NSTIFS

B-3 Data Tables Summary
NTAM NTAB NTAW NTAP NTAMT NGCP

B-4 Gravitational Acceleration
GRAV

B-5 Buckling Mode Imperfections
WIMPFA IMSTEP IMMODE IMRUN

B-6 Inertial Loads Summary
ISYS IA IOM IAL IOPT

B-6a Inertial Loads — Acceleration
AX AY AZ
Appendix A

B-6b Inertial Loads — Angular Velocity
  OMX  OMY  OMZ

B-6c Inertial Loads — Angular Acceleration
  ALX  ALY  ALZ

B-6d Inertial Loads — Position Vector
  X  Y  Z

Discretization and Connectivity Summary

F-1 Grid Summary
  ( NROWS(i) NCOLS(i) ), i = 1, NUNITS

F-2 Stiffener Summary
  IUNIT  NRGS  NSTR

G-1 Shell Unit Connections
  MUNIT  MBOUND  NUNIT  NBOUND  NDEFS  INC1  INC2  INC3  INC4

G-2 Partial Displacement Compatibility
  IU1  IR1  IC1  ID1  IU2  IR2  IC2  ID2  ND1A  ND1B  INC1  ND2A  ND2B  INC2  NDEFS

G-2a Partial Compatibility Incrementations
  JU1  JR1  JC1  JD1  JU2  JR2  JC2  JD2

G-2c Partial Displacement Compatibility
  IU1  IR1  IC1  IL1  ID1  IU2  IR2  IC2  IL2  ID2  ND1A  ND1B  INC1  ND2A  ND2B  INC2

G-2d Partial Compatibility Looping and Incrementations
  NDEFS  JU1  JR1  JC1  JL1  JD1  JU2  JR2  JC2  JL2  JD2

G-3 Constraint—Record 1
  NTERMS  NX  INC1  INC2  INC3  INC4  INC5

G-4 Constraint—Record 2
  IU(i)  IX(i)  IY(i)  ID(i)  CC(i)  IZ(i)

G-5 Crack Inserted Node Set—Record 1
  NCRACK  INNODS  INELTS  ITEAR  ACRIT  CTOD
  ACRITT  SAWL  SAWT  CSCALE  IDREC
G-6 Crack Inserted Node Set—Record 2

ICOPEN ICUNIT ICROW1 ICCOL1 ICROW2 ICCOL2
   JCUNIT JCROW1 JCCOL1 JCROW2 JCCOL2 THETA

G-7 Crack Inserted Node Set—Record 3

JETYPE JCUNIT JCROW1 JCCOL1 JCROW2 JCCOL2

H-1 Element Unit Summary (NUNITE records required)

NUPT NT1 NT2 NT3 NT4 NT5 IUWP IUWE IUDIMP IUWLE NS5

Data Tables

I-1 Material Properties

ITAM NESP IPLST ITANST ICREEP IPLANE

I-2 Material Elastic Properties

E1 U12 G RHO A1 E2 A2

I-3 Material Plastic Properties

( E(i), S(i) ), i=1, NESP

I-3a Material Creep Properties

ACO BCO M N

I-4a Mount Element Table Size

IMNT NRD NRV

I-4b Relative Displacement Vector

DISP(j), j = 1, NRD

I-4c Relative Velocity Vector

RVEL(i), i = 1, NRV

I-4d Mount Force Matrix

FORCE(j), j = 1, NRD

I-5a GCP Command Record

COMMAND INFO(j), j = 1, 7
I-6a  Linear Elastic Isotropic GCP Material
   \[ E \quad \text{GNU} \quad \rho \quad \alpha \quad \beta \quad T \quad M \]

I-7a  Linear Elastic Orthotropic GCP Material
   \[ E_1 \quad E_2 \quad E_3 \quad G_{12} \quad G_{13} \quad G_{23} \quad P_{12} \quad P_{13} \quad P_{23} \quad \rho \quad A_1 \quad A_2 \quad A_3 \quad B_1 \quad B_2 \quad B_3 \quad T \quad M \]

I-8a  Plane-Strain-Plasticity GCP Material
   \[ E \quad \text{GNU} \quad \rho \quad \alpha \quad \beta \quad T \quad M \]

I-9a  Mechanical Sublayer Plasticity GCP Material
   \[ E \quad \text{GNU} \quad \rho \quad \alpha \quad \text{NSUBS} \quad T \]

I-9b  Stress-Strain Curve for a Given State
   \[ (E(i), S(i)), \quad i = 1, \text{NSUBS} \]

I-10a Linear Orthotropic Elastic Brittle GCP Material
   \[ E_1 \quad E_2 \quad E_3 \quad G_{12} \quad G_{13} \quad G_{23} \quad P_{12} \quad P_{13} \quad P_{23} \quad \rho \quad A_1 \quad A_2 \quad A_3 \quad B_1 \quad B_2 \quad B_3 \quad T \quad M \quad \varepsilon_{1c} \quad \varepsilon_{1t} \quad \varepsilon_{2c} \quad \varepsilon_{2t} \quad \varepsilon_{6f} \quad \varepsilon_{3c} \quad \varepsilon_{3t} \quad \varepsilon_{4f} \quad \varepsilon_{5f} \quad \chi \quad \chi_0 \quad \chi_1 \quad \varepsilon_{List} \quad \alpha \quad \beta \quad \text{IFAIL} \quad \text{IDGRD} \quad \text{VISF0} \quad \text{VISF1} \quad \text{VISFF} \quad \text{VISFF} \quad \text{DEPHI0} \quad \text{DELTA} \quad \text{DMIN} \quad \beta \quad \text{ETA} \quad \text{BETAC} \]

I-11a PDLAM GCP Material
   \[ E_1 \quad E_2 \quad E_3 \quad G_{12} \quad G_{13} \quad G_{23} \quad P_{12} \quad P_{13} \quad P_{23} \quad \rho \quad A_1 \quad A_2 \quad A_3 \quad B_1 \quad B_2 \quad B_3 \quad T \quad M \quad \chi \quad \chi_0 \quad \chi_1 \quad \text{VISF0} \quad \text{VISF1} \quad \text{VISFF} \quad \text{DEPHI0} \quad \text{DELTA} \quad \text{DMIN} \quad \beta \quad \text{ETA} \quad \text{BETAC} \]

I-12a ABAQUS UMAT GCP Material
   \[ \text{PROPS}(1) \quad \text{PROPS}(2) \ldots \quad \text{PROPS}(40) \]

I-13a SHM-Membrane GCP Material
   \[ E \quad \text{GNU} \quad \rho \quad \alpha \quad \beta \quad T \quad M \quad \text{PENLTY} \quad \text{IWRINK} \quad \text{ISTATE} \]

I-14a Nonlinear Orthotropic Elastic GCP Material
   \[ E_1 \quad E_2 \quad E_3 \quad G_{12} \quad G_{13} \quad G_{23} \quad P_{12} \quad P_{13} \quad P_{23} \quad \rho \quad A_1 \quad A_2 \quad A_3 \quad B_1 \quad B_2 \quad B_3 \quad T \quad M \quad S_{6666} \]

I-21a GCP Shell Fabrication Record
   \[ \text{MATID}(j) \quad j = NX, \text{NLAYER} \]

I-21b GCP Shell Integration Points Record
   \[ \text{INTSHL}(j) \quad j = NX, \text{NLAYER} \]

I-21c GCP Shell Layer Thickness Record
   \[ \text{THKSHL}(j) \quad j = NX, \text{NLAYER} \]
I-21d  GCP Shell Layer Orientation Record
   ANGSHL(j)  j = NX, NLAYER

I-21e  Shear Factor Specification Record
   SCF1  SCF2

I-22a  GCP Solid Fabrication Record
   THICK

I-22b  GCP Solid Fabrication Orientation Record
   ANGLE

J-1   Cross-Section
   ITAB  KCROSS  MATB  NSUB  TORJ  SCY  SCZ  NSOYZ  KAPY  KAPZ

J-2a  General Cross-Section—Record 1
   BA  BIY  BIZ  BIYZ

J-2b  General Cross-Section—Record 2
   ( SOY(i), SOZ(i) ), i = 1, NSOYZ

J-3a  General Subelement Cross-Section
   SA(i)  SY(i)  SZ(i)  SIY(i)  SIZ(i)  SIYZ(i)  ISP(i)

J-3b  Rectangular Subelement Cross-Section
   Y1(i)  Y2(i)  Z1(i)  Z2(i)  ISOC(i)

J-4a  Arbitrary Cross-Section—Record 1
   BMA

J-4b  Arbitrary Cross-Section—Record 2
   ( CCC(i,j), j=1,4 ), i=1,4

K-1   Shell Wall Properties
   ITAW  KWALL  NLAY  NLIP  NSMRS  SHEAR1  SHEAR2

K-2   Layered Wall
   MATL  TL  ZETL  LSOL
Appendix A

K-3a  Fiber Reinforced Wall—Record 1  
      MATF  MATM

K-3b  Fiber Reinforced Wall—Record 2  
      TT XX ZETW O

K-4a  Corrugation Stiffened Wall—Record 1  
      MATC  MATS  CT  CC  CH  CD  CB

K-4b  Corrugation Stiffened Wall—Record 2  
      TS  PHI  ANC

K-5a  General Wall—Record 1  
      TA  MAT  ITVS

K-5b  General Wall—Record 2  
      (CCC(i,j), j = 1, 6), i = 1, 6

K-5c  General Wall—Record 3  
      (CTS(i,j), j = 1, 2), i = 1, 2

K-6  Smeared Stiffener  
      ICROSM  SPASM  ZETSM  XSISM  ECZSM

L-1  User Parameters Summary  
      NPI  NPF

L-2a  Integer Parameters  
      USERINT(i), i = 1, NPI

L-2b  Floating-Point Parameters  
      USERFLO(i), i = 1, NPF
Shell Units

M-1 Shell Type
ISHELL IGLOBE NROWS NCOLS NLAYS NFABS

M-2 Shell Surface Constants
PROP(i), i=1,8

M-3 Shell Unit Orientation—Straight-Line Boundary
NREP NCEP XGEP YGEP ZGEP

M-4a Shell Unit Orientation—Corner Point 1
XGC1 YGC1 ZGC1

M-4b Shell Unit Orientation—Corner Point 2
XGC2 YGC2 ZGC2

M-4c Shell Unit Orientation—Corner Point 3
XGC3 YGC3 ZGC3

M-4d Shell Unit Orientation—Translation
XG YG ZG

M-4e Shell Unit Orientation—Rotation
XGROT YGROT ZGROT

M-5 Shell Wall (NFABS records required)
IWALL IWIMP ZETA ECZ ILIN IPLAS IRAMP

M-6 Shell Imperfections
X1 Y1 XL YL WAMP ID

M-7a Random Imperfection Shapes
N1 N2 NI M1 M2 MI KTEST

M-7b Random Imperfection Amplitudes
GRAMP RANK EXP

N-1 Discretization Control
KELT NNX NNY IRREG IUGRID INTEG IPENL MESH1 MESH2 MESH3 MESH4 KELTX
N-2  X-Segment Length
    SEGLX(i), i=1,NNX

N-3  X-Segment Spacing
    NSEGX(i), i=1,NNX

N-4  X-Coordinate
    X(i), i = 1, NRWS

N-5  Y-Segment Length
    SEGLY(j), j = 1, NNY

N-6  Y-Segment Spacing
    NSEGY(j), j = 1, NNY

N-7  Y-Coordinate
    Y(j), j = 1, NCLS

N-8  Mesh Irregularity
    NRW1  NRW2  NCL1  NCL2

O-1a  Discrete Ring—Record 1
    ICROSS  XSI  ECY  ECZ  ILIN  IPLAS

O-1b  Discrete Ring—Record 2
    IR  JR1  JR2  XR  YR1  YR2

O-2a  Discrete Stringer—Record 1
    ICROSS  XSI  ECY  ECZ  ILIN  IPLAS

O-2b  Discrete Stringer—Record 2
    JS  IS1  IS2  YS  XS1  XS2

P-1  Boundary Conditions—Record 1
    IBLN(i), i = 1, 4  IBOND

P-2  Boundary Conditions—Record 2
    ITRA  IROT
P-3 Boundary Conditions—Record 3
JBLN(i), i = 1, 4

P-4 Boundary Conditions—Record 4
JTRA JROT

Q-1 Loads Summary
NSYS NICS NAMS NUSS NHINGE NMOMNT NLEAST IPRESS

Q-2 Load Set Summary
ISYS NN IFLG

Q-3 Load Definition
P LT LD LI LJ LAX NX INC1 INC2 INC3 ILAY

Q-4 Attached Mass (NN records required)
GM IRM ICM LAYER NX INC1 INC2 INC3

Q-5 Uniform Stress State for Eigenanalysis
PNXA PNYA PNXYA PNXB PNYB PNXYB

Q-6 Cable Hinge Restraint
IROW ICOL HRU HRV HRW LAYER NX INC1 INC2 INC3

Q-7 Cable Hinge Moment
IROW ICOL MSYS RUM RVM RWM LAYER NX INC1 INC2 INC3

Q-8a Least Squares Loading Summary
NSQR IUNIT IROW ICOL SCALE

Q-8b Least Squares Load Definition
P LU LR LC LNDA LNDB LNDINC

R-1 Output Control—Record 1
IPRD IPRR IPRS IPRF NSELD NSELS IPRDSP IPRSTR ISL ISS ISD

R-2 Output Control—Record 2
IROWD ICOLD
R-3 Output Control—Record 3

IROWS ICOLS

Element Units—Points & Lines

S-1 User Points (upts protocol)

IUPT IUS IRS ICS XG YG ZG IUVW IRUVW IAUX NPTS ILAY

S-1a User Point Incrementations (upts protocol)

JUPT JUS JRS JCS Dxg Dyg Dzg

S-2 Auxiliary Coordinate System (upts protocol)

XAX XAY XAZ YAX YAY YAZ

S-3 User Points (user-points protocol)

IUPT IUS IRS ICS XG YG ZG IUVW IRUVW IAUX NPTS ILAY

S-3a User Point Incrementations (user-points protocol)

JUPT JUS JRS JCS Dxg Dyg Dzg

S-4 Auxiliary Coordinate System (user-points protocol)

XAX XAY XAZ YAX YAY YAZ

S-5 Contact-Line Definition

LINEID UNITID NRECS

S-5a Contact Line on a Shell Unit

II JJ NPTS INC

S-5b Contact Line on an Element Unit

II NPTS INC

Element Units—Edef Protocol

T-1 Spring Element

N1 N2 N3 KELT NX INC1 INC2 INC3 USERELT INC4

T-1a Mount

IMNT1 IMNT2 RLX1 RLY1 RLZ1 RLX2 RLY2 RLZ2
T-1b Rigid or Soft Link
  SCALE

T-1c Generalized Fastener
  IMNT1 IMNT2 IMNT3 IMNT4 IMNT5 IMNT6 PLAS1 PLAS2 PLAS3 PLAS4 PLAS5 PLAS6

T-2 Beam
  N1 N2 N3 KELT ICROSS XSI ECY ECZ ILIN IPLAS NX USERELT

T-2a Beam Incrementations
  INC1 INC2 INC3 INC4

T-3 Triangular Shell
  N1 N2 N3 KELT IWALL ZETA ECZ ILIN IPLAS IANG USERELT

T-3a Wall Reference Vector
  RX RY RZ

T-4 Quadrilateral Shell
  N1 N2 N3 N4 KELT IWALL ZETA ECZ ILIN IPLAS INTEG IPENL IANG USERELT

T-4a Extra Nodes
  NODE(i), i=5,n

T-4b Wall Reference Vector
  RX RY RZ

T-5 Contact, Sandwich, and Solid Element Flags
  N810 N820 N822 N830 N840 N845 N847 N849 N880

T-6 E810 PAD Contact Element
  N1 N2 N3 N4 N5 N6 N7 N8 KELT ITAB OFFSET NX USERELT

T-6a PAD Element Incrementations
  I1 I2 I3 I4 I5 I6 I7 I8 I9

T-7 General Contact Definition
  KELT NSRF NPTS

T-7a E820 Row & Column Contact-Element Specifications
  USRF TYPE LI LJ ID NI NJ
Appendix A

T-7b E820 Element-Number Contact-Element Specifications
    USRF  TYPE  I1  I2  INC  ID

T-7c Row & Column Contact-Point Specifications
    UNITP  LI  LJ  RADIUS  TOUCHE  NI  NJ

T-7d Point-Number Contact-Point Specifications
    UNITP  I1  I2  INC  RADIUS  TOUCHE

T-8 Line-Contact Interaction Definition
    LINE1  LINE2  IPEN  NX  INC1  INC2  INC3

T-9 E830 6-Node Sandwich Element Definition
    KELT  ILIN  INTEG  IPEN  NX  NY  USERC  USER1  USER2

T-9a E830 Lower Face-Sheet Properties
    N1  N2  N3  IFABL  ZETAL  ECZL  IPLASL  IANGL

T-9b E830 Lower Face-Sheet Wall Reference Vector
    RXL  RYL  RZL

T-9c E830 Upper Face-Sheet Properties
    N4  N5  N6  IFABU  ZETAU  ECZU  IPLASU  IANGU

T-9d E830 Upper Face-Sheet Wall Reference Vector
    RXU  RYU  RZU

T-9e E830 Core Properties
    IFABC  ZETAC  IPLASC  IANGC

T-9f E830 Core Reference Vector
    RXC  RYC  RZC

T-9g E830 X-Direction Incrementations
    I1  I2  I3  I4  I5  I6  I7  I8  I9

T-9h E830 Y-Direction Incrementations
    J1  J2  J3  J4  J5  J6  J7  J8  J9

T-10 E840 8-Node Sandwich Element Definition
    KELT  ILIN  INTEG  IPEN  NX  NY  USERC  USER1  USER2
T-10a E840 Lower Face-Sheet Properties
   N1 N2 N3 N4 IFABL ZETAL ECZL IPLASL IANGL

T-10b E840 Lower Face-Sheet Wall Reference Vector
   RXL RYL RZL

T-10c E840 Upper Face-Sheet Properties
   N5 N6 N7 N8 IFABU ZETAU ECZU IPLASU IANGU

T-10d E840 Upper Face-Sheet Wall Reference Vector
   RXU RYU RZU

T-10e E840 Core Properties
   IFABC ZETAC IPLASC IANGC

T-10f E840 Core Reference Vector
   RXC RYC RZC

T-10g E840 X-Direction Incrementations
   I1 I2 I3 I4 I5 I6 I7 I8 I9 I10 I11

T-10h E840 Y-Direction Incrementations
   J1 J2 J3 J4 J5 J6 J7 J8 J9 J10 J11

T-11 E845 10-Node Sandwich Transition Element Definition
   KELT ILIN INTEG IPEN IEDGE NX NY USER

T-11a E845 Lower Face-Sheet Properties
   N1 N2 N3 N4 N5 IFABL ZETAL ECZL IPLASL IANGL

T-11b E845 Lower Face-Sheet Wall Reference Vector
   RXL RYL RZL

T-11c E845 Upper Face-Sheet Properties
   N6 N7 N8 N9 N10 IFABU ZETAU ECZU IPLASU IANGU

T-11d E845 Upper Face-Sheet Wall Reference Vector
   RXU RYU RZU

T-11e E845 Core Properties
   IFABC ZETAC IPLASC IANGC
Appendix A

T-11f E845 Core Reference Vector
   RXC  RYC  RZC

T-11g E845 X-Direction Incrementations
   I1  I2  I3  I4  I5  I6  I7  I8  I9  I10  I11

T-11h E845 Y-Direction Incrementations
   J1  J2  J3  J4  J5  J6  J7  J8  J9  J10  J11

T-12 E847 14-Node Sandwich Transition Element Definition
   KELT  ILIN  INTEG  IPEN  IEDGE  NX  NY  USER

T-12a E847 Lower Face-Sheet Properties
   N1  N2  N3  N4  N5  N6  N7  IFABL  ZETAL  ECZL  IPLASL  IANGL

T-12b E847 Lower Face-Sheet Wall Reference Vector
   RXL  RYL  RZL

T-12c E847 Upper Face-Sheet Properties
   N8  N9  N10  N11  N12  N13  N14  IFABU  ZETAU  ECZU  IPLASU  IANGU

T-12d E847 Upper Face-Sheet Wall Reference Vector
   RXU  RYU  RZU

T-12e E847 Core Properties
   IFABC  ZETAC  IPLASC  IANGC

T-12f E847 Core Reference Vector
   RXC  RYC  RZC

T-12g E847 X-Direction Incrementations
   I1  I2  I3  I4  I5  I6  I7  I8  I9  I10  I11  I12  I13  I14  I15

T-12h E847 Y-Direction Incrementations
   J1  J2  J3  J4  J5  J6  J7  J8  J9  J10  J11  J12  J13  J14  J15

T-13 E849 18-Node Sandwich Element Definition
   KELT  ILIN  INTEG  IPEN  NX  NY  USERC  USER1  USER2

T-13a E849 Lower Face-Sheet Properties
   N1  N2  N3  N4  N5  N6  N7  N8  N9  IFABL  ZETAL  ECZL  IPLASL  IANGL
T-13b E849 Lower Face-Sheet Wall Reference Vector
   RXL  RYL  RZL

T-13c E849 Upper Face-Sheet Properties
   N10 N11 N12 N13 N14 N15 N16 N17 N18  IFABU  ZETAU  ECZU  IPLASU  IANGU

T-13d E849 Upper Face-Sheet Wall Reference Vector
   RXU  RYU  RZU

T-13e E849 Core Properties
   IFABC  ZETAC  IPLASC  IANGC

T-13f E849 Core Reference Vector
   RXC  RYC  RZC

T-13g E849 X-Direction Increments
   ( IX(k), k=1,18 ) IXC IX1 IX2

T-13h E849 Y-Direction Increments
   ( IY(k), k=1,18 ) IYC IY1 IY2

T-14 E880-Family Solid Element
   KELT  IFAB  IANG  ILIN  IPLAS  NX NY

T-14a E880 Solid Element Nodes
   ( NODE(k), k=1,NPTS ) USERELT

T-14b E880 X-Direction Increments
   IX1 IX2 IX3 IX4 IX5 IX6 IX7 IX8  IUX

T-14c E880 Y-Direction Increments
   IY1 IY2 IY3 IY4 IY5 IY6 IY7 IY8  IUY

T-14d E880 Material Orientation Record
   XFX XFY XFZ  YFX YFY YFZ
Element Units—Ecom Protocol

T-100  Element Command Record
Ecom Control or Element-Specification Record

T-110  Additional E110 Elements
N1 N2 N3 KELT NX INC1 INC2 INC3 USERELT INC4

T-110a E110 Element Data
IMNT1 IMNT2 RLX1 RLY1 RLZ1 RLX2 RLY2 RLZ2

T-120  Additional E120 Elements
N1 N2 N3 KELT NX INC1 INC2 INC3 USERELT INC4

T-120a E120 Element Data
SCALE

T-121  Additional E121 Elements
N1 N2 N3 KELT NX INC1 INC2 INC3 USERELT INC4

T-121a E121 Element Data
SCALE

T-130  Additional E130 Elements
N1 N2 N3 KELT NX INC1 INC2 INC3 USERELT INC4

T-130a E130 Element Data
IMNT1 IMNT2 IMNT3 IMNT4 IMNT5 IMNT6 PLAS1 PLAS2 PLAS3 PLAS4 PLAS5 PLAS6

T-210  Additional E210 Elements
N1 N2 N3 KELT ICROSS XSI ECY ECZ ILIN IPLAS NX USERELT

T-210a E210 Incrementations
INC1 INC2 INC3 INC4

T-250  Additional E250 Elements
N1 N2 N3 KELT ICROSS XSI ECY ECZ ILIN IPLAS NX USERELT

T-250a E250 Incrementations
INC1 INC2 INC3 INC4
T-320  Additional E320 Elements
 N1 N2 N3  KELT  IWALL  ZETA  ECZ  ILIN  IPLAS  IANG  USERELT  NX NY

T-320a  E320 X-Direction Incrementations
 IX1  IX2  IX3  IX4

T-320b  E320 Y-Direction Incrementations
 IY1  IY2  IY3  IY4

T-320c  E320 Wall Reference Vector
 RX  RY  RZ

T-330  Additional E330 Elements
 N1 N2 N3  KELT  IWALL  ZETA  ECZ  ILIN  IPLAS  IANG  USERELT  NX NY

T-330a  E330 X-Direction Incrementations
 IX1  IX2  IX3  IX4

T-330b  E330 Y-Direction Incrementations
 IY1  IY2  IY3  IY4

T-330c  E330 Wall Reference Vector
 RX  RY  RZ

T-410  Additional E410 Elements
 N1 N2 N3 N4  KELT  IWALL  ZETA  ECZ  ILIN  IPLAS  INTEG  IPENL  IANG  USERELT  NX NY

T-410a  E410 X-Direction Incrementations
 IX1  IX2  IX3  IX4  IX5

T-410b  E410 Y-Direction Incrementations
 IY1  IY2  IY3  IY4  IY5

T-410c  E410 Wall Reference Vector
 RX  RY  RZ

T-411  Additional E411 Elements
 N1 N2 N3 N4  KELT  IWALL  ZETA  ECZ  ILIN  IPLAS  INTEG  IPENL  IANG  USERELT  NX NY

T-411a  E411 X-Direction Incrementations
 IX1  IX2  IX3  IX4  IX5
Appendix A

T-411b E411 Y-Direction Incrementations
  IY1  IY2  IY3  IY4  IY5

T-411c E411 Wall Reference Vector
  RX  RY  RZ

T-480  Additional E480 Elements
  N1  N2  N3  N4  KELT  IWall  ZETA  ECZ  ILIN  IPLAS  INTEG  IPENL  IANG  USERELT  NX  NY

T-480a E480 Extra Nodes Specification
  N5  N6  N7  N8  N9

T-480b E480 X-Direction Incrementations
  IX1  IX2  IX3  IX4  IX5  IX6  IX7  IX8  IX9  IXU

T-480c E480 Y-Direction Incrementations
  IY1  IY2  IY3  IY4  IY5  IY6  IY7  IY8  IY9  IYU

T-480d E480 Wall Reference Vector
  RX  RY  RZ

T-510  Additional E510 Elements
  N1  N2  N3  N4  KELT  IWall  ZETA  ECZ  ILIN  IPLAS  INTEG  IPENL  IANG  USERELT  NX

T-510a E510 Extra Node Specification
  N5

T-510b E510 Incrementations
  IX1  IX2  IX3  IX4  IX5  IXU

T-510c E510 Wall Reference Vector
  RX  RY  RZ

T-710  Additional E710 Elements
  N1  N2  N3  N4  KELT  IWall  ZETA  ECZ  ILIN  IPLAS  INTEG  IPENL  IANG  USERELT  NX

T-710a E710 Extra Nodes Specification
  N5  N6  N7

T-710b E710 Incrementations
  IX1  IX2  IX3  IX4  IX5  IX6  IX7  IXU
Appendix A

T-710c E710 Wall Reference Vector
   RX  RY  RZ

T-810 E810 PAD Contact Element
   N1  N2  N3  N4  N5  N6  N7  N8  KELT  ITAB  OFFSET  NX  USERELT

T-810a PAD Element Incrementations
   I1  I2  I3  I4  I5  I6  I7  I8  I9

T-820 General Contact Definition
   KELT  NSRF  NPTS

T-820a E820 Row & Column Contact-Element Specifications
   USRF  TYPE  LI  LJ  ID  NI  NJ

T-820b E820 Element-Number Contact-Element Specifications
   USRF  TYPE  I1  I2  INC  ID

T-820c Row & Column Contact-Point Specifications
   UNITP  LI  LJ  RADIUS  TOUCHE  NI  NJ

T-820d Point-Number Contact-Point Specifications
   UNITP  I1  I2  INC  RADIUS  TOUCHE

T-822 Line-Contact Interaction Definition
   LINE1  LINE2  IPEN  NX  INC1  INC2  INC3

T-830 E830 6-Node Sandwich Element Definition
   KELT  ILIN  INTEG  IPEN  NX  NY  USERC  USER1  USER2

T-830a E830 Lower Face-Sheet Properties
   N1  N2  N3  IFABL  ZETAL  ECZL  IPLASL  IANGL

T-830b E830 Lower Face-Sheet Wall Reference Vector
   RXL  RYL  RZL

T-830c E830 Upper Face-Sheet Properties
   N4  N5  N6  IFABU  ZETAU  ECZU  IPLASU  IANGU

T-830d E830 Upper Face-Sheet Wall Reference Vector
   RXU  RYU  RZU
T-830e  E830 Core Properties
IFABC  ZETAC  IPLASC  IANGC

T-830f  E830 Core Reference Vector
RXC  RYC  RZC

T-830g  X-Direction Incrementations
I1  I2  I3  I4  I5  I6  I7  I8  I9

T-830h  Y-Direction Incrementations
J1  J2  J3  J4  J5  J6  J7  J8  J9

T-840  E840 8-Node Sandwich Element Definition
KELT  ILIN  INTEG  IPEN  NX  NY  USERC  USER1  USER2

T-840a  E840 Lower Face-Sheet Properties
N1  N2  N3  N4  IFABL  ZETAL  ECZL  IPLASL  IANGL

T-840b  E840 Lower Face-Sheet Wall Reference Vector
RXL  RYL  RZL

T-840c  E840 Upper Face-Sheet Properties
N5  N6  N7  N8  IFABU  ZETAU  ECZU  IPLASU  IANGU

T-840d  E840 Upper Face-Sheet Wall Reference Vector
RXU  RYU  RZU

T-840e  E840 Core Properties
IFABC  ZETAC  IPLASC  IANGC

T-840f  E840 Core Reference Vector
RXC  RYC  RZC

T-840g  X-Direction Incrementations
I1  I2  I3  I4  I5  I6  I7  I8  I9  I10  I11

T-840h  Y-Direction Incrementations
J1  J2  J3  J4  J5  J6  J7  J8  J9  J10  J11

T-845  E845 10-Node Sandwich Transition Element Definition
KELT  ILIN  INTEG  IPEN  IEDGE  NX  NY  USER
T-845a E845 Lower Face-Sheet Properties
   N1 N2 N3 N4 N5 IFABL ZETAL ECZL IPLASL IANGL

T-845b E845 Lower Face-Sheet Wall Reference Vector
   RXL RYL RZL

T-845c E845 Upper Face-Sheet Properties
   N6 N7 N8 N9 N10 IFABU ZETAU ECZU IPLASU IANGU

T-845d E845 Upper Face-Sheet Wall Reference Vector
   RXU RYU RZU

T-845e E845 Core Properties
   IFABC ZETAC IPLASC IANGC

T-845f E845 Core Reference Vector
   RXC RYC RZC

T-845g E845 X-Direction Incrementations
   I1 I2 I3 I4 I5 I6 I7 I8 I9 I10 I11

T-845h Y-Direction Incrementations
   J1 J2 J3 J4 J5 J6 J7 J8 J9 J10 J11

T-847 14-Node Sandwich Transition Element Definition
   KELT ILIN INTEG IPEN IEDGE NX NY USER

T-847a E847 Lower Face-Sheet Properties
   N1 N2 N3 N4 N5 N6 N7 IFABL ZETAL ECZL IPLASL IANGL

T-847b E847 Lower Face-Sheet Wall Reference Vector
   RXL RYL RZL

T-847c E847 Upper Face-Sheet Properties
   N8 N9 N10 N11 N12 N13 N14 IFABU ZETAU ECZU IPLASU IANGU

T-847d E847 Upper Face-Sheet Wall Reference Vector
   RXU RYU RZU

T-847e E847 Core Properties
   IFABC ZETAC IPLASC IANGC
Appendix A

T-847f  E847 Core Reference Vector
          RXC  RYC  RZC

T-847g  X-Direction Incrementations
           I1  I2  I3  I4  I5  I6  I7  I8  I9  I10  I11  I12  I13  I14  I15

T-847h  E847 Y-Direction Incrementations
           J1  J2  J3  J4  J5  J6  J7  J8  J9  J10  J11  J12  J13  J14  J15

T-849  E849 18-Node Sandwich Element Definition
          KELT  ILIN  INTEG  IPEN  NX  NY  USERC  USER1  USER2

T-849a  E849 Lower Face-Sheet Properties
          N1  N2  N3  N4  N5  N6  N7  N8  N9  IFABL  ZETA1  ECZL  IPLASL  IANGL

T-849b  E849 Lower Face-Sheet Wall Reference Vector
          RXL  RYL  RZL

T-849c  E849 Upper Face-Sheet Properties
          N10  N11  N12  N13  N14  N15  N16  N17  N18  IFABU  ZETAU  ECZU  IPLASU  IANGU

T-849d  E849 Upper Face-Sheet Wall Reference Vector
          RXU  RYU  RZU

T-849e  E849 Core Properties
          IFABC  ZETAC  IPLASC  IANGC

T-849f  E849 Core Reference Vector
          RXC  RYC  RZC

T-849g  X-Direction Incrementations
          ( IX(k), k=1,18 )  IXC  IX1  IX2

T-849h  Y-Direction Incrementations
          ( IY(k), k=1,18 )  IYC  IY1  IY2

T-860  E860 ORACLE Solid Element
          KELT  NNODES  IFAB  IANG  ILIN  IPLAS  NX  NY

T-860a  E860 Solid Element Nodes
          ( NODE(k), k=1,NNODES )  USERELT
Appendix A

T-860b E860 X-Direction Incrementations
( IX(k), k=1,NNODES ) IXU

T-860c E860 Y-Direction Incrementations
( IY(k), k=1,NNODES ) IYU

T-860d E860 Material Orientation Record
XFX XFY XFZ YFX YFY YFZ

T-881 E881 8-Node Solid Element
KELT IFAB IANG ILIN IPLAS NX NY

T-881a E881 Solid Element Nodes
( NODE(k), k=1,8 ) USERELT

T-881b E881 X-Direction Incrementations
IX1 IX2 IX3 IX4 IX5 IX6 IX7 IX8 IXU

T-881c E881 Y-Direction Incrementations
IY1 IY2 IY3 IY4 IY5 IY6 IY7 IY8 IYU

T-881d E881 Material Orientation Record
XFX XFY XFZ YFX YFY YFZ

T-882 E882 18-Node Solid Element
KELT IFAB IANG ILIN IPLAS NX NY

T-882a E882 Solid Element Nodes
( NODE(k), k=1,18 ) USERELT

T-882b E882 X-Direction Incrementations
( IX(k), k=1,18 ) IXU

T-882c E882 Y-Direction Incrementations
( IY(k), k=1,18 ) IYU

T-882d E882 Material Orientation Record
XFX XFY XFZ YFX YFY YFZ

T-883 E883 27-Node Solid Element
KELT IFAB IANG ILIN IPLAS NX NY
Appendix A

T-883a E883 Solid Element Nodes
   ( NODE(k), k=1,27 ) USERELT

T-883b E883 X-Direction Incrementations
   ( IX(k), k=1,27 ) IXU 151

T-883c E883 Y-Direction Incrementations
   ( IY(k), k=1,27 ) IYU 152

T-883d E883 Material Orientation Record
   XFX XFY XFZ YFX YFY YFZ

T-885 E885 20-Node Solid Element
   KELT IFAB IANG ILIN IPLAS NX NY

T-885a E885 Solid Element Nodes
   ( NODE(k), k=1,20 ) USERELT

T-885b E885 X-Direction Incrementations
   ( IX(k), k=1,20 ) IXU

T-885c E885 Y-Direction Incrementations
   ( IY(k), k=1,20 ) IYU

T-885d E885 Material Orientation Record
   XFX XFY XFZ YFX YFY YFZ

T-900 E9XX User-Defined Element
   KELT ID IFAB IANG ILIN IPLAS NX NY NZ

T-900a E9XX User-Element Nodes
   ( NODE(k), k = 1, NNODES )

T-900b User-Element X-Direction Incrementations
   ( IX(k), k = 1, NNODES ) IXU

T-900c User-Element Y-Direction Incrementation
   ( IY(k), k = 1, NNODES ) IYU

T-900d User-Element Z-Direction Incrementations
   ( IZ(k), k = 1, NNODES ) IZU
T-900e  User-Element Material Orientation Vector
   XFX XFY XFZ

T-900f  User-Element Material Orientation Vectors
   XFX XFY XFZ   YFX YFY YFZ

T-928  E928 3-Node Curved Beam UEL
   KELT   ID   IFAB IANG ILIN IPLAS   NX

T-928a E928 UEL Nodes
   N1  N2  N3  N4

T-928b E928 UEL X-Direction Incrementations
   IX1  IX2  IX3  IX4   IXU

T-928c E928 UEL floatVariables
   Area  Iy  Iz  J  Material  ShearFactorY  ShearFactorZ  ECC(1)  ECC(2)  SCC(1)  SCC(2)

T-940  E940 MIN4 Quadrilateral UEL
   KELT   ID   IFAB IANG ILIN IPLAS   NX  NY

T-940a E940 UEL Nodes
   N1  N2  N3  N4

T-940b E940 UEL X-Direction Incrementations
   IX1  IX2  IX3  IX4   IXU

T-940c E940 UEL Y-Direction Incrementations
   IY1  IY2  IY3  IY4   IYU

T-940d E940 UEL Material Orientation Vector
   XFX XFY XFZ

T-940e E940 UEL Material Orientation Vectors
   XFX XFY XFZ   YFX YFY YFZ

T-940f E940 UEL floatVariables
   UniformPressure(1)  UniformPressure(2)

T-940g E940 UEL integerVariables
   IntegOrder  LoadType
Element Units—Loadings, *et al.*

U-1  Loads Summary
NSYS NICS NAMS NUSS NHINGE NMOMNT NLEAST IPRESS

U-2  Load Set Summary
ISYS NN IFLG

U-3  Load Definition (NN records required)
P LI LT LD LT LJ LAX NDEFS INC1 INC2 INC3

U-4  Attached Mass
GM NM NDEFS INC

U-5  Uniform Stress State for Eigenanalysis
PNXA PNYA PNXYA PNXB PNYB PNXYB

U-6  Cable Hinge Restraint
IHND HRU HRV HRW NDEFS INC

U-7  Cable Hinge Moment
IMND MSYS RUM RVM RWM NDEFS INC

U-8a  Least Squares Loading Summary
NSQR IUNIT IROW ICOL SCALE

U-8b  Least Squares Load Definition
P LU LR LC LNDA LNDB LNDINC

V-1  Output Control—Record 1
IPRD IPRR IPRE IPRS IPRP IPRF NSELD NSELS IPRDSP IPRSTR ISL ISS ISD

V-2  Output Control—Record 2
INOD1 INOD2 INODI

V-3  Output Control—Record 3
IELS1 IELS2 IELSI
W-1  Linear-Stiffness Contribution—Record 1
   NRDOF  NRNOD  NRKIJ  KLSTF  NRDIS  NRFOR

W-2a  Linear-Stiffness Contribution—Record 2a
   IUNIT  IROW  ICOL   JUNIT  JROW  JCOL

W-2b  Linear-Stiffness Contribution—Record 2b
   \{( KIJ(m, n), n=1,NRDOF ), m=1,NRDOF \}

W-2c  Linear-Stiffness Contribution—Record 2b
   IUNIT(n)  IROW(n)  ICOL(n)  n = 1,2,3, ..., NRNOD
Solution Input—*BIN* File

Summary and Control Parameters

A-1 Case Title
   COMMENT

B-1 Analysis Type Definition
   INDIC IPOST ILIST ICOR IMPTHE ICHIST IFLU ISOLVR NFABC

B-2 Solver Options
   ICPACT ITER IPRIM IPRIS ISAVE

B-3 Gradient Fabrication Specification Records
   KFABTP KFB

Computational Strategy Parameters

C-1 Load Multipliers
   STLD(1) STEP(1) FACM(1) STLD(2) STEP(2) FACM(2) ITEMP NFIX

C-3 Nonlinear Stress State
   NLDS IXEV

C-4 Load Factors
   PLDS(i), i=1,NLDS

C-5 Suppress Selected Freedoms
   IFIX(i), i=1,NFIX

D-1 Strategy Parameters
   ISTART NSEC NCUT NEWT NSTRAT DELX WUND

D-2 Eigenvalue Control
   NSEC DELEV IPRINT

D-3 Cluster Definition
   NEIG SHIFT EIGA EIGB
E-1 Time Integration—Record 1
   TMIN TMAX DT SUP ALPHA BETA GAMMA THOLD

E-2 Time Integration—Record 2
   IMPL METHOD IERRF IVELO IFORCE IPA IPB

E-3 Load History—Record 1
   CA1 CA2 CA3 CA4 CA5 CA6

E-4 Load History—Record 2
   CB1 CB2 CB3 CB4 CB5 CB6

E-5 Weighted Modal Initial Velocity
   EIGA IMSTEP IMMODE IMRUN

ET-1 Solution Control
   NPATH NEV NSOL IE NFIX LDMAX IUPLDA IUPLDB

The Equivalence Transformation Bifurcation Processor (ET)

ET-1 Solution Control
   NPATH NEV NSOL IE NFIX LDMAX IUPLDA IUPLDB
Appendix B

STAGS 5.0 User Manual April, 2009

B

STAPL Input

Introduction

STAPL is a stand-alone STAGS post-processor that generates PostScript- or pdf-formatted plots of a complete STAGS model (or a partial STAGS model) in its undeformed or deformed states—with contour maps of user-selected displacement, strain, stress,... fields that are superimposed on them as and if appropriate. The original version of STAPL was developed some years ago as an in-house processor to generate platform-independent, PostScript-formatted plots for use in verifying STAGS models and for use in visualizing and understanding results obtained by STAGS for those models. That version of STAPL was updated to construct PostScript or pdf-formatted output files soon after Adobe’s PDF technology was developed—and has been updated several times since then. The most recent enhancements to STAPL have brought it as up-to-date as possible with respect to the current version of the STAGS program. STAPL is distributed with the STAGS program suite.

Input file

To use the STAPL post-processor, the analyst must first generate a STAGS model by creating a case.inp input file for it and running STAGS’ s1 program to a successful conclusion with that input file. STAPL uses the case.sav file that s1 produces as its primary source of information about the geometry and construction of the model for which one or more plots are to be generated. The analyst who wants to use STAPL to generate plots that show the deformed model—with or without contours showing the displacement, strain, stress, or any other relevant field information—must first obtain those results by running STAGS’ s2 program successfully.
Execution and input requirements

To execute STAPL on a UNIX system—for a particular case—the user must navigate to a “working” directory that must contain information that is required to make any plots and may contain other information that may be required or used advantageously to make the plots that the user wants to generate for that case. Positioned in that working directory, the user must then execute STAPL via the following OS-level command:

```
% stapl case
```

where case is the name of the case.

If the analyst only wants STAPL to produce one or more plots that show one or more parts of the original (undeformed) model, the only file that must be included in the working directory is the case.sav file that s1 has generated for the case in question. Other files and scripts may be present in the working directory, but the case.sav file must be included. If the analyst wants STAPL to produce one or more plots that show deformations, strains, stresses and/or other field-variable “results” (and/or imperfections and/or eigensolutions), s2’s case.res (results) and case.rst (indexing) files must also be in his working directory—along with s2’s case.egv (eigensolutions) file and/or s1’s case.imp (imperfections) file as and if those files are needed.

In either case, the OS-level stapl case command initiates execution of STAPL for the case in question. After its initialization operations, the first thing that STAPL does is check to determine whether or not the analyst’s working directory contains an optional input (text) file called case.pin.

If the working directory does not contain a case.pin file, then STAPL operates in its interview mode. In this mode, STAPL prints messages specifying the information that it needs and waits for the user to respond with appropriate answers before continuing. Operating in its interview mode, STAPL takes user responses into account and only requests input data that are appropriate for the case and plotting situation at hand. Operating in this mode, STAPL generates an annotated case.pin output file that the user can edit and employ with subsequent (batch-mode) STAPL runs to generate the same kinds of plots with other model components and/or results. The user who wants to generate a large number of plots should start by executing STAPL in its interview mode—then use his (or her) favorite text editor to modify the case.pin file generated in that run to construct new case.pin files for batch-mode executions of STAPL to generate the additional plots that are wanted. The ins and outs of
STAPL’s interview mode are beyond the scope of this appendix. The interested user is strongly encouraged to read the “STAPL User Manual” for more information about that.

If the working directory does contain a case.pin file, then STAPL operates in its batch mode—using that case.pin file (and case.sav and other case.* files, as appropriate) to generate the plot(s) that case.pin specifies.

The case.pin input requirements for STAPL are described in the pages that follow the next three paragraphs.

Output files

As noted above, a successful interview-mode execution of STAPL gives the user an annotated case.pin output text file that can be edited and used for subsequent batch-mode executions of the program. With any execution of STAPL, the program also generates an informative (but disposable) case.pout text file that summarizes what STAPL does with the user’s case.pin, case.sav and case.* results data.

When STAPL is executed successfully, it generates a one-page PostScript-formatted file for each plot that the analyst requests, or a single pdf-formatted file that contains all of the plots that STAPL generates during that execution. The output format is specified by the analyst, as described in the PL-2 record, on page B-5.

The STAPL post-processor does not modify the case.sav, case.res, case.rst, case.egv, case.imp or any other case.* files that it may need and use to produce the requested plots.
**PL-1 Case Title**

The first thing that STAPL expects (and reads) from the user’s live or scripted input stream is a plotting-case-title character string called **TITLE**—which may contain up to but not more than 72 alphanumeric characters. STAPL prints the user’s **TITLE** string on each plot that it produces.

After this Case Title record, the user can and is encouraged to append comment strings and to add comment lines to his (or her) input stream to document that input. Comment strings can be included at the end of any data line by interrupting or terminating the data with a “$” character and using the remainder of that line for comment material. Comment lines can be included within or between data records by starting those lines with “c” or “$” as the first non-blank character. STAPL prints a listing of the complete input file—including any comment records—at the beginning of the *case.pout* text output file that STAPL produces for the current execution of the program.

---

**TITLE**

**TITLE** case title (72 characters or less)

**Note:** If the User’s **TITLE** specification has the 4-character ‘3DMF’ string imbedded within it, STAPL will produce a *case.3dmf* output file for use with the VIEWER application that is being developed for examining and doing other exciting things with STAGS models and solutions.

---

* go to PL-2
**PL-2 Post-Processing Summary Record**

This record specifies the number of plots that the analyst wants **STAPL** to generate, the output format that the analyst wants, and other information that **STAPL** needs before it can generate the plot(s) that the analyst desires.

---

NPLOT IPREP IPRS KDEV KXSTEP

**NPLOT**
number of plots to be generated (see the **Note**, below)

**IPREP**
solution-data-required flag:
0 - model and solution archive data are both required (default)
1 - model data required; solution data not required

**IPRS**
undeformed-model display flag:
0 - show the undeformed model (as a dashed blue wire frame) in addition to the deformed model (default)
1 - show the deformed model only

**KDEV**
output-format flag:
0 - generate all plots (one plot per page) in a single, Acrobat (pdf-formatted) *case.pdf* output file (default)
1 - generate each plot \( k \) as a separate, PostScript-formatted *case.k.ps* output file

**KXSTEP**
solution-scale-factor index:
0 - determine scale factors for each solution step independently of all other steps (default)
\( > 0 \) - use scale factors from step \# **KXSTEP** for each solution step treated in the current **STAPL** execution

---

**Note:** A **STAPL** input (**PIN**) file may contain more than one PL-2 record, for convenience. All of the input records in the **PIN** file are read and echoed to the **POUT** output file, but **STAPL** will only process and generate plots for the very first PL-2 record in the **PIN** file.
PL-3 Plot Description Record

This record and the records it requires generates one plot. \textbf{N\textsc{p}lot} PL-3 records are required.

\begin{verbatim}
KPLOT VIEW ITEM STEP MODE IFRNG COLOR ICOMP LAYER FIBR LAY3D FACE

\textbf{KPLOT} primary plot control flag:
\begin{itemize}
\item 0 - plot all or a portion of the undeformed model (default)
\item -1 - plot all or a portion of the undeformed model, coloring the outer surface red and the inner surface blue
\item 1 - plot all or a portion of the deformed model, with a selected solution projected as its “deformed geometry”
\item 2 - plot all or a portion of the deformed model, with a selected continuous solution field texture-mapped onto the deformed surface in color or in grayscale mode
\item 3 - same as 2, when the solution is discontinuous
\item -3 - plot contours using a continuous solution field, for elements of the same type—even if their fabrications are different
\item 4 - plot gradient contours on all or a portion of the deformed model, with a selected continuous gradient field texture-mapped onto the deformed surface in color or in grayscale mode
\item 5 - same as 4, when the gradient field is discontinuous
\item 6 - reserved for use with \textsc{Stags’ Viewer} application
\end{itemize}

\textbf{VIEW} plot view control flag:
\begin{itemize}
\item 0 - plot the entire model (default)
\item >0 - plot only the \textbf{N\textsc{un}it} units listed in record PL-4, described below
\item <0 - plot only the fabrications listed in record PL-4, described below
\end{itemize}

\textbf{ITEM} secondary plot control flag:
\begin{itemize}
\item when \textbf{KPLOT} = 0 — number-plotting option:
\begin{itemize}
\item \textbf{ITEM} = 0 - do not plot the element, fabrication or node numbers
\item \textbf{ITEM} = 1 - plot \textsc{Stags’} global element number inside each element
\item \textbf{ITEM} = 2 - plot the User’s element number inside each element
\item \textbf{ITEM} = 3 - plot the unit number and the user’s element number
\item \textbf{ITEM} = 4 - only plot the unit number inside each element
\item \textbf{ITEM} = 5 - plot the fabrication number in each element
\item \textbf{ITEM} = 6 - plot \textsc{Stags’} global node number beside each node
\item \textbf{ITEM} = 7 - plot the global node numbers with minimum font size
\end{itemize}
\end{itemize}
\end{verbatim}
ITEM = 8 - plot the User’s node number beside each node
ITEM = 9 - same as 8, for wire-frame plot(s) of the model
ITEM = 10 - highlight beam elements (with red color and wide lines)

when KPLOT = 1, 2 or 3 or -3 — solution-type flag:
ITEM = 1 - plot a displacement solution
ITEM = 2 - plot a velocity vector
ITEM = 3 - plot an imperfection vector
ITEM = 4 - plot an eigenvector
ITEM = -4 - plot a displacement-differences vector (see MODE, below)

when KPLOT = 2 or 3 or -3 — solution-type flag:
ITEM = 5 - plot stress resultants (see ICOMP, below):

for 2D elements and sandwich-element face sheets:
   $Nx, Ny, Nxy, Mx, My, Mxy$

for 3D elements: ignored

ITEM = 6 - plot strains/curvatures (integrated) (see ICOMP, below):

for 2D elements and sandwich-element face sheets:
   $Ex, Ey, Exy, Kx, Ky, Kxy$

for sandwich-element core components:
   $Exx, Eyy, Eyz, Ezx, Exy$

for 3D elements:
   $Exx, Eyy, Ezz, Syz, Szx, Sxy$ (in the fabrication system)

ITEM = 7 - plot stresses ($sigf, sigm, sigp$), (strains, plane-strains)
(see ICOMP, below):

for 2D elements and sandwich-element face sheets:
   $Sx, Sy, Sxy$

for sandwich-element core components:
   $Sxx, Syy, Szz, Syz, Szx, Sxy$

for 3D elements:
   $Sxx, Syy, Szz, Syz, Szx, Sxy$ (in the fabrication system)

ITEM = 8 - plot ply damage percentage (when ICOMP=0, as noted below)
ITEM = 8 - plot plastic strains (when ICOMP=1–3)

ITEM = 9 - plot energy densities
ITEM = 10 - plot temperature gradients
ITEM = 11 - plot element thicknesses
ITEM = 12 - plot internal forces (not operational, yet)
ITEM = 13 - plot external forces (not operational, yet)

ITEM = 14 - plot PFA or UMAT state variable (ICOMP = 1–20)
ITEM = 15 - plot selected ply failure types (see the PL-3c record, below)

STEP load step number (for static analysis, 0 selects the linear solution; for transient analysis, 0 selects the initial-conditions)

MODE vector selection parameter:
if ITEM = 4, MODE = eigenvector (mode) number;
if ITEM = –4, displacement difference from step MODE to step STEP

IFRNG fringe-plot control flag:
0 - use 32 color or grayscale gradations (default)
1 - minimal (16-color) gradation
2 - full fringe plot

COLOR color/scale-control flag:
0 - generate color plot, with maximum to minimum scale
1 - generate color plot, with minimum to maximum scale
2 - generate grayscale plot, with maximum black & minimum white
3 - generate grayscale plot, with maximum white & minimum black

ICOMP tertiary plot control flag (meaningful only when KPLOT = 2 or 3):
0 - plot the solution magnitude (when 1 ≤ ITEM ≤ 4, above), or
plot the effective stress (when ITEM = 7) (see note, below), or
plot the ply damage percentage (when ITEM = 8)

1 - plot U or NX or EXI or S1F or EPX or EXX or SXX — depending on ITEM, above

2 - plot V or NY or EYI or S2F or EPY or EYY or SYY — depending on ITEM

3 - plot W or NXY or EXYI or T12F or EPXY or EZZ or SZZ — depending on ITEM
4 - plot RU or MX or KX or S1M or EYZ or SYZ — depending on ITEM

5 - plot RV or MY or KY or S2M or EZX or SZX — depending on ITEM

6 - plot RW or MXY or 2KXY or T12M or EXY or SXY — depending on ITEM

7 - plot the surface normal component QX or EXZ
    plot S1P (principal stress) (when ITEM = 5 or 6), or
    plot QX or EXZ
    plot S1P (principal stress) (when ITEM = 7)

8 - plot QY or EYZ
    plot S2P (principal stress) (when ITEM = 5 or 6), or
    plot QY or EYZ
    plot S2P (principal stress) (when ITEM = 7)

9 - plot N1P (principal resultant) (when ITEM = 5), or
    plot S3P (principal stress) (when ITEM = 7)

10 - plot N2P (principal resultant) (when ITEM = 5), or
    plot mxshr (E840 maximum shear) (when ITEM = 7)

11 - plot N3P (principal resultant) (when ITEM = 5), or
    plot EX (total strain at selected fiber location) (when ITEM = 7)

12 - plot EY (total strain at selected fiber location) (when ITEM = 7)

13 - plot EXY (total strain at selected fiber location) (when ITEM = 7)

14 - plot S3F (plane strain at selected fiber location) (when ITEM = 7)

Note: with ITEM = 7, ICOMP = 0, selects effective stresses in the fabrication (1–3), material (4–6) or principal (7–9) systems

LAYER selection flag for 2D shell-element layer (used when ITEM = 7 or 8):

0 - plot layer # 1 (inner layer), if FIBR < 2
   plot layer # NLAY (outer layer), if FIBR = 2
> 0 - plot layer # LAYER (omitted if LAYER > NLAY)

FIBR selection flag for 2D shell-element layer fiber (used when ITEM = 7, 8 or 10):

0, 1 - plot the inner fiber (minimum z value)
2 - plot the outer fiber (maximum z value)
-1 - plot the inner and outer fibers
LAY3D, FACE  layer and face selection flags for sandwich or solid elements:

For E840-sandwich core elements (when face elements are not plotted):

- LAY3D = 0 - plot all sandwich core elements (when FACE = -1 or -2)
- LAY3D > 0 - plot sandwich core element # LAY3D (in stack)
- FACE = -1 - plot E840 core elements, omitting their E410 face sheets
- FACE = -2 - E840 core elements, omitting all other elements

For E840-sandwich face elements:

- LAY3D - this item is ignored when E840 face elements are plotted
- FACE = 0 - plot all of the E410 (inner and outer) face sheets
- FACE = 1 - only plot the inner face sheets
- FACE = 2 - only plot the outer face sheets

For E88* solid elements (when ITEM = 6 or 7):

- LAY3D = 0 - plot all layers of solid elements
- LAY3D > 0 - only plot layer # LAY3D

- FACE = 0 - plot all of the solid-element surfaces
- FACE > 0 - only plot surface # FACE (where 1 ≤ FACE ≤ 6)

if ( KPLOT = 4 or 5 ) then
  go to PL-3a
elseif ( KPLOT = 1 and ITEM = 15 ) then
  go to PL-3c
elseif ( VIEW ≠ 0 ) then
  go to PL-4
else
  go to PL-5
endif

Note: For convenience, a STAPL input (PIN) file may contain more than NPLOTS PL-3 records—where NPLOTS is the value that the user specifies for the N PLOT parameter in the very first PL-2 record in the PIN file. STAPL reads all of the records in the user’s PIN file (and prints them in its POUT file), but STAPL will only process and generate the first NPLOTS plots.
PL-3a Strain Gradient Control Record

This record is read if and only if the KPLOT parameter is 4 or 5 on the PL-3 record.

<table>
<thead>
<tr>
<th>MODE1</th>
<th>MODE2</th>
<th>MODINC</th>
</tr>
</thead>
<tbody>
<tr>
<td>field type or starting index:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 - displacement gradient or load vector (in which case the values of MODE2 and MODINC are ignored)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt; 0 - first mode number</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| MODE2 | last mode number (must not exceed 24 for shell) |

| MODINC | mode-number increment |

if ( MODE1 > 0 ) then
  go to PL-3b
else
  go to PL-4
endif
PL-3b Strain Gradient Coefficient Record

( STGD(i), i = MODE1, MODE2, MODINC )

STGD

list of strain gradient coefficients (note: coefficients less than 25 not in this list are set equal to zero)

if ( [ KPLOT = 2 or 3 or -3 ] and ITEM = 15 ) then
go to PL-3c
else
  go to PL-4
endif
PL-3c Selected Ply Failure Record

This record is read if and only if the KPLOT parameter is 2 or 3 and the KPLOT parameter is 15 on the PL-3 record.

<table>
<thead>
<tr>
<th>MATID</th>
<th>FLAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATID</td>
<td>material identifier</td>
</tr>
<tr>
<td></td>
<td>0 - plot all elements fabricated with ply-failure-analysis materials</td>
</tr>
<tr>
<td></td>
<td>&gt; 0 - plot elements with material # MATID</td>
</tr>
<tr>
<td>FLAG</td>
<td>failure-mode selection flag:</td>
</tr>
<tr>
<td></td>
<td>0 - all failure modes</td>
</tr>
<tr>
<td></td>
<td>1 - fiber tensile failure mode</td>
</tr>
<tr>
<td></td>
<td>2 - matrix tensile failure mode</td>
</tr>
<tr>
<td></td>
<td>3 - inplane shear failure mode (shell elements), or interlaminal tensile failure mode (solid elements)</td>
</tr>
<tr>
<td></td>
<td>4 - transverse (13) failure mode</td>
</tr>
<tr>
<td></td>
<td>5 - transverse (23) failure mode</td>
</tr>
<tr>
<td></td>
<td>6 - inplane shear failure mode (solid elements)</td>
</tr>
<tr>
<td></td>
<td>-1 - fiber compressive failure mode</td>
</tr>
<tr>
<td></td>
<td>-2 - matrix compressive failure mode</td>
</tr>
<tr>
<td></td>
<td>-3 - interlaminal compressive failure mode (solid elements)</td>
</tr>
</tbody>
</table>

if ( VIEW ≠ 0 ) then
    go to PL-4
else
    go to PL-5
endif
This record is read if and only if the plot view control flag (VIEW) on record PL-3 is not zero. With the PL-4 record, the analyst can select a subset of the shell and/or element units in the complete model to be plotted in the current STAPL plot (when VIEW > 0) or a subset of the element–fabrication types to be plotted (when VIEW < 0). The magnitude of VIEW must clearly be less than or equal to the total number of units in the complete model (when VIEW > 0) or the number of fabrications (when VIEW < 0).

\[
\begin{align*}
  ( \text{LIST}(j), \ j=1, \ |\text{VIEW}| ) \\
\end{align*}
\]

**LIST(j)** unit number for shell or element unit to be included in the current plot (when VIEW > 0), or fabrication number to be included (when VIEW < 0)

\[\text{go to PL-5}\]
**PL-5 Plot Control Record**

This record is required for specification of scaling, orientation, and other parameters for the current STAPL plot.

<table>
<thead>
<tr>
<th>DSSCALE</th>
<th>NROTS</th>
<th>LWSCALE</th>
<th>RNGMIN</th>
<th>RNGMAX</th>
<th>NUMBRS</th>
<th>NFONT</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSSCALE</td>
<td>NROTS</td>
<td>LWSCALE</td>
<td>RNGMIN</td>
<td>RNGMAX</td>
<td>NUMBRS</td>
<td>NFONT</td>
</tr>
</tbody>
</table>

**DSSCALE**
scaling factor to be used with solution data, for deformed-model plots:

- 0 - **STAPL** sets DSSCALE = 0.5 (default value) and scales the displacement vector by multiplying it by DSSCALE times the maximum dimension of the projected model
- > 0 - **STAPL** scales the displacement vector by multiplying it by DSSCALE times the maximum dimension of the projected model
- < 0 - **STAPL** scales the displacement vector by multiplying it by |DSSCALE| times the maximum dimension of the projected mode; the actual deformations are shown if DSSCALE = -1.0

**NROTS**
number of PL-6 orientation records required:

- 0 - **STAPL** uses its default orientation parameters
- > 0 - **STAPL** reads and uses orientation parameters specified on the NROTS type PL-6 records following this record

**LWSCALE**
factor to be used in scaling line width (element edges):

- 0 - use the default line width
- > 0 - set the line width to LWSCALE times the default width
- - 1 - omit edge lines, except on unit boundaries
- - 2 - add wire frame plot of the undeformed model (dashed blue lines)

**RNGMIN**
plot–range scaling factor (see RNGMAX)

**RNGMAX**
plot–range scaling factor:

If RNGMIN = RNGMAX = 0, the range scale for the plot uses the minimum and maximum values from the solution vector; if RNGMIN and/or RNGMAX are not zero, the range scale uses RNGMIN and RNGMAX for its extrema
NUMBRS annotation flag, for contour plots:

0 - do not plot element, node or fabrication numbers
1 - plot STAGS’ global element number inside each element
2 - plot the User’s element number inside each element
3 - plot the unit numbers and the User’s element numbers
4 - only plot the unit numbers on the model
5 - only plot the fabrication numbers on the model
6 - plot STAGS’ global node numbers on the model
7 - plot STAGS’ global node numbers (minimum font size)
8 - plot the User’s node numbers on the model
9 - plot the User’s node numbers on a wire-frame model

NFONT font-size specification flag (for element or node numbers):

0 - use STAPL’s default font size
> 0 - use a font size determined by the NFONT parameter, where:
  the font size increases with increasing NFONT and where
  NFONT is in the range \(1 \leq \text{NFONT} \leq 16\)

```plaintext
if ( NROTS > 0 ) then
  go to PL-6
else
  follow instructions at end of PL-6
endif
```
PL-6 Orientation Records

A record of this type is required for specification of each user-specified orientation (rotation) operation that STAPL is to perform. Rotations are performed in the order specified by these records. **NROTS** (PL-5) PL-6 records must be included here. For more information about this type of orientation specification, see the “B-1b Sequence of Model Rotations” description on 5-11, in Chapter 5 of this document.

**IROT ROT**

**IROT** axis selection flag:
1 - rotate the model by **ROT** degrees about its current X axis
2 - rotate the model by **ROT** degrees about its current Y axis
3 - rotate the model by **ROT** degrees about its current Z axis

**ROT** rotation angle, in degrees

**NPLOT** (PL-2) number of plots to be generated

if ( less than **NPLOT** plots have been generated ) then
    return to **PL-3**
else
    data deck is complete
endif
 Examples

An interview-mode execution of STAPL for the ACORO test-suite case, with simple “show me the model” user responses to STAPL prompts, generated the plot shown in Figure B.1:

![STAPL plot of undeformed ACORO configuration](image)

Figure B.1  STAPL plot of undeformed ACORO configuration

STAPL also generated the following acoro.pin output text file for this execution:

```
STAPL Example
  1 1 0 1 $PL-2  NPLLOT,IPREP,IPRS,KDEV
  0 0 $PL-3  KPLLOT,NUNIT
  0.00000000E+00 0 $PL-5  DSscale,NROTS
```

This case.pin file shows (with NROTS=0) that the analyst did not specify the spatial orientation of the model in this interview-mode run. The plot that STAPL generated used the program’s default rotation angles to define the orientation of the model and its default color sequence to show each shell unit in the model in a different color.
Seeing that STAPL’s default orientation hides significant parts of the structure, the user might want to edit the acoro.pin file that STAPL generated in the interview-mode execution, to specify more suitable rotation angles, as follows:

```
STAPL Example -- Reoriented configuration
1 1 0 1          $ PL-2  NPLOT,IPREP,IPRS,KDEV
0 0          $ PL-3  KPLOT,NUNIT
0.00000000E+00 3 $ PL-5  DSCALE,NROTS
1  -80.0           $ PL-6
2  -25.0           $ PL-6
3  -15.0           $ PL-6
```

A batch-mode execution of STAPL after doing that produces the configuration plot that is shown in Figure B.2:

![STAPL plot of reoriented ACORO model](image)

Figure B.2  STAPL plot of reoriented ACORO model

Encouraged by this success, the intrepid user might modify this second acoro.pin file to use this improved orientation and other specifications to generate a third plot that shows how this model is deformed at step 8 of the analysis that he/she has performed—generating the following acoro.pin file for the next batch-mode STAPL run:
STAPL Example -- W Displacements at Step 8

<table>
<thead>
<tr>
<th>Step</th>
<th>Displacement</th>
<th>Contours</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-80.0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-25.0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-15.0</td>
<td></td>
</tr>
</tbody>
</table>

Note that this user has chosen to show the \( w \) displacement contours in full color on the deformed configuration. **STAPL** produces the plot shown in Figure B.3 with this `acoro.pin` file:

![STAPL plot of ACORO results for step 8](image)

Figure B.3  **STAPL** plot of ACORO results for step 8

Note that the deformed shape shown here has the actual displacements scaled by a factor of 6.518—the (0.05 times the maximum dimension of the projected model) value that **STAPL** computed when the user set `DSCALE` equal to zero.
C

PITRANS Input

Introduction

PITRANS is a stand-alone post-processor that was developed years ago to translate STAGS model and solution data for a particular (user-specified) “case” into formatted input files for PATRAN and IDEAS programs. PITRANS is rarely (if ever) used for that purpose, these days; but it is used frequently to generate PATRAN-formatted “neutral” files and/or IDEAS-formatted “universal” files that contain model and solution information that can be extracted from those files and used for other purposes. The PITRANS post-processor is distributed with the STAGS program suite.

Prerequisites

To use PITRANS to translate any STAGS data for the case at hand into PATRAN- or IDEAS-formatted files, the analyst must first generate a STAGS model by creating a case.inp input file for that case and by running STAGS’ s1 program successfully with that input file. PITRANS uses s1’s case.sav output file as its sole source of information about the geometry and construction of the user’s model. To use PITRANS to generate PATRAN- or IDEAS-formatted output files that contain or depend upon STAGS’ solution data, the analyst must also create one or more case.bin input files for STAGS’ s2 program and run s2 successfully with those input files to perform the analyses that they request. PITRANS uses s2’s case.res and case.rst output files as its primary sources of information about the solution(s) that STAGS has generated for the case at hand.

Execution of the program

To execute PITRANS on a LINUX or on a UNIX system, the user must navigate to a “working” directory that must contain all of the “STAGS” files that PITRANS needs to generate the
PATRAN- or IDEAS-compatible output file(s) that he wants. Most frequently, that working
directory must contain s1’s case.sav file and s2’s case.res and case.rst files for the case at
hand. It may also contain other files, but they are not required and are generally ignored.
Positioned in that directory, the user may then execute PITRANS via one or the other of the
following two OS-level commands:

% pitrans

or

% pitrans < case.pix

where case is the name of the case at hand and where case.pix is a text-formatted input
file that the user must have created before executing PITRANS.

Input requirements and output files

The PITRANS program (like STAPL) operates in what we call its “interview” mode—where
it prompts the user for an answer and performs one or more actions in response to that
answer—then prompts the user again (and again, and again) until the user is finished
specifying what he wants the program to do. Unlike the STAPL program, PITRANS does not
construct a “journal” file that records the user’s answers and does not test to determine if a
program-generated or a user-constructed journal (or script) file exists and use it (instead of
the user’s keyboard responses) to read the user’s input for the case at hand. The
adventurous user can “force” PITRANS to operate in a “batch” mode (of sorts) by
constructing a bare-bones (or a very carefully annotated) text file that contains the answers
that PITRANS expects (and little if anything else) and by executing PITRANS as suggested
in the second OS-level command, above—redirecting the program to seek the answers that
it expects for the questions that it asks from that file instead getting it from the keyboard.
The file name for that “input” file is arbitrary. We are calling it case.pix here and in the
remainder of this Appendix for convenience and consistency.

In any event, PITRANS is fundamentally an interview-mode program. PITRANS prompts the
user to give it the input that it needs, checks the user’s input to determine that it is valid
(and prompts him for it again if it is not), asks for more information (if necessary), and
generates the output that the user requests—following the logic that is shown in the
“navigation” diagram on the following page. The RED items in that “flow” chart represent
“addresses” (resembling statement numbers in FORTRAN and C programs). The BLUE items
represent prompts that direct the user to give PITRANS the input information (data record)
that it needs to do what the user wants it to do, followed by the program’s attempt to read
the user’s answers to those prompts. The upper-case BLACK items in that diagram are the
PX1  specify the Output Format (PX-1 record ) FORM
PX2  specify the Stags Case Name (PX-2 record ) CASE
PX3  specify the Output Data Option (PX-3 record ) ODATA

if ( ODATA = 'M' or 'm' ) then
go to PX7
elseif ( ODATA = 'V' or 'v' ) then
go to PX4
elseif ( ODATA = 'I' or 'i' ) then
go to PX5
elseif ( ODATA = 'S' or 's' ) then
go to PX8
elseif ( ODATA = 'Q' or 'q' ) then
    finished !!!
else
go to PX3
endif

PX4  specify the Vector Type (PX-4 record ) VTYPE

if ( VTYPE = 'D' or 'd' or 'V' or 'v' or 'F' or 'f' ) go to PX7
    go to PX4

PX5  specify the Integrated Stress Type (PX-5 record ) STYPE

if ( STYPE = 'S' or 's' or 'M' or 'm' ) go to PX6
    go to PX5

PX6  specify the Next Step (PX-6 record ) ISTEP

if ( ISTEP = 'Q' ) go to PX3
    go to PX6

PX7  specify the Model Type (PX-7 record ) MTYPE

if ( MTYPE ≠ 'F' or 'f' or 'C' or 'c' ) go to PX7
if ( ODATA = 'M' or 'm' ) go to PX3
if ( ODATA = 'V' or 'v' ) go to PX8

PX8  specify the Next Step (PX-7 record ) STEP

if ( STEP = 'Q' ) go to PX3
if ( ODATA = 'S' or 's' ) go to PX9
    go to PX8

PX9  specify the Layer Number (PX-8 record ) LAYER

if ( LAYER is not valid ) go to PX9
if ( nr840 > 0 or nr880 > 0 ) then

PX9a specify the Output Point Location (PX-8a record ) SLOC

if ( SLOC ≠ 'C' or 'c' or 'I' or 'i' or 'N' or 'n' ) go to PX9a
    endif
if ( plasx and .not. makeplas ) then

PX9b specify Plastic Strains (PX-8b record ) IPLAST

    endif

PX9c specify Reference Frame (PX-8c record ) FRAME

    go to PX8
user’s answers (the user-specified input parameters)—with the ‘X’ and ‘x’ items indicating the answers that PITRANS recognizes for the prompt at hand. The lower-case black items in that diagram are parameters that PITRANS extracts from the STAGS database for the case at hand or constructs using those parameters and the user’s responses to prompts that PITRANS has made during the current execution.

PITRANS starts (at PX1 in the navigation chart) by asking the user to specify the “output format” that he wants the program to use for all of the output files that it generates during the current run—by printing the following “prompt” to the screen:

   Enter output type: (P)ATRAN, (I)DEAS = >

and waiting (where the ‘>’ symbol is) for the user to specify (in the FORM input parameter) that he wants PITRANS to give him output in PATRAN’s “neutral file” format (in which case he should answer the prompt with an upper case ‘P’ or a lower case ‘p’ response) or to specify that he wants PITRANS to give him output in IDEAS’ “universal file” format (in which case he should answer the prompt with an upper case ‘I’ or a lower case ‘i’ response). If the user’s answer is anything other than one or the other of those four characters, PITRANS will repeat the PX1 prompt and wait for the user’s next answer to the same question.

Immediately after getting a valid answer to its first prompt, PITRANS responds (at PX2 in the navigation chart) by asking the user for the STAGS “case” name for the case at hand—by printing the following “prompt” to the screen:

   Enter Stags Case Name = >

and waiting (where the ‘>’ symbol is) for the user’s answer to that question (the CASE input parameter) at the PX2 location. PITRANS expects a character string here that identifies the case at hand and enables PITRANS to gain access to the case.sav (and other) STAGS database files in the user’s working directory to extract and process the information that the user is about to request.

Then (at location PX3 in the flow chart) PITRANS continues by asking the user what he wants the program to do next—by printing the following “prompt” to the screen:

   Data: (M)odel only data (nodes, elements, bc, etc) (V)ector results (displ, veloc, accel, forces) (I)ntegrated Stresses -or- Mid-Surface Strains (S)tress and Strains (equivalent nodal values) (Q)uit making Stags plot files ................ = >

and waiting (where the ‘>’ symbol is) for the user to specify (via the ODATA input parameter) that he wants PITRANS to give him information about the model (with an ‘M’ or ‘m’ response to the prompt), or to give him displacement, velocity, or reaction force results from the solution database (with a ‘V’ or ‘v’ response), or to give him integrated stress or mid-surface strain results from the solution database (with an ‘I’ or ‘i’ response), or to give him nodal stress or
strain results (with an ‘S’ or ‘s’ response), or to “quit” the current execution of the program (with a ‘Q’ or ‘q’ response). If the user’s answer is anything other than one or the other of those ten characters, PITRANS will repeat the PX3 prompt and wait for the user’s next answer to the same question. If the user’s answer is one of those ten characters (one of those five options), PITRANS will react in accord with the user’s request (as explained below).

If the user sets ODATA = ‘M’ or ‘m’ at its PX3 control point—to ask for information about the model—PITRANS will go to its PX7 control point and ask the user to specify that the STAGS database for the case at hand contains information for the full model (by setting MTYPE = ‘F’ or ‘f’) or to specify that it contains information for a condensed model (by setting MTYPE = ‘C’ or ‘c’). If the user’s answer to the PX7 prompt is anything other than one or the other of those four characters, PITRANS will repeat the prompt and wait for the user’s next answer to the same question. If the answer is one of those four characters, PITRANS will give the user a formatted output file called case.mdl that contains the desired information and will then return to its PX3 control point and prompt the user for his next instruction(s).

If the user sets ODATA = ‘V’ or ‘v’ at the program’s PX3 control point—to ask for displacement, velocity, or reaction force results from the solution database—PITRANS will go to its PX4 control point and ask the user to specify (via the VTYPE parameter) that he wants the program to give him displacement results (by setting VTYPE = ‘D’ or ‘d’), or velocity results (by setting VTYPE = ‘V’ or ‘v’) or reaction forces (by setting VTYPE = ‘F’ or ‘f’). If the user’s answer is anything other than one or the other of those six characters, PITRANS will repeat the PX4 prompt and wait for the user’s next answer to the same question. If the user’s answer is one of those six characters, PITRANS will transfer to its PX7 control point and ask the user to specify that the STAGS database for the case at hand contains information for the full model (by setting MTYPE = ‘F’ or ‘f’) or that it contains information for a condensed model (by setting MTYPE = ‘C’ or ‘c’). If the user’s answer to this PX7 prompt is anything other than one or the other of those four characters, PITRANS will repeat the prompt and wait for the user’s next answer to the same question. If the answer is one of those four characters, PITRANS will transfer to its PX8 control point, will display its current “load table” (which contains some or all of the load step, load factor, time, and “# of modes” information that the user needs to know for the case at hand) on the screen and save it in a formatted (text) output file called case.step (if it hasn’t already done so), and will prompt the user to specify (via the STEP input parameter) what he wants the program to do:

If the user has set STEP = ‘P’ or ‘S’ at PX8, then PITRANS will print information from another part of the “load table,” then return to PX8 to ask the user what he wants the program to do next.

If the user has set STEP = N at PX8 (where N is a valid step number for the case at hand), then PITRANS will extract the (displacement, velocity or reaction force) results that he wants (for that step number) from the STAGS solutions database
and will give it to him in a vector-formatted output file called \textit{case.dis.N} or \textit{case.vel.N} or \textit{case.fint.N} (depending on \texttt{VTYPE}). After doing that, \texttt{PITRANS} will return to its \texttt{PX8} control point to prompt the user for his next instruction(s).

If the user has set \texttt{STEP = 'Q'} at \texttt{PX8}, then \texttt{PITRANS} will “quit” its \texttt{PX8} activities and will return to the program’s \texttt{PX3} control point to prompt the user for his next instruction(s).

If the user sets \texttt{ODATA = 'I' or 'i'} at the program’s \texttt{PX3} control point—to ask for \textit{integrated stress} or \textit{mid-surface strain} results from the solution database—\texttt{PITRANS} will go to its \texttt{PX5} control point and prompt the user to specify that he wants \texttt{PITRANS} to give him integrated stresses (resultants and moments) (by setting the \texttt{STYPE} input parameter equal to \texttt{'S' or 's'}) or that he wants \texttt{PITRANS} to give him mid-surface strains and curvatures (by setting \texttt{STYPE = 'M' or 'm'}). If the user’s answer is anything other than one or the other of those four characters, \texttt{PITRANS} will repeat the \texttt{PX5} prompt and wait for the user’s next answer to the same question. If the user’s answer is one of those four characters, \texttt{PITRANS} will then go to its \texttt{PX6} control point, display its current “load table” on the screen and save it in a formatted (text) output file called \textit{case.step} (if it hasn’t already done so), and prompt the user to specify (via the \texttt{ISTEP} input parameter) what he wants the program to do next:

If the user has set \texttt{ISTEP = 'P' or 'S'} at \texttt{PX6}, then \texttt{PITRANS} will print information from another part of the “load table,” then return to \texttt{PX6} to ask the user what he wants the program to do next.

If the user has set \texttt{ISTEP = N} at \texttt{PX6} (where \texttt{N} is a valid step number for the case at hand) and if he has set \texttt{STYPE = 'S' or 's'} at \texttt{PX5}, then \texttt{PITRANS} will extract the integrated stresses that he wants for step \texttt{N} from the \texttt{STAGS} solutions database and give them to him in a formatted output file called \textit{case.elmI.N}—or if he has set \texttt{STYPE = 'M' or 'm'} at \texttt{PX5}, then \texttt{PITRANS} will extract the midsurface strains and curvatures from the database and give them to him in \textit{case.elmR.N}. After doing one or the other of those two things, then \texttt{PITRANS} will return to its \texttt{PX6} control point to prompt the user for his next instruction(s).

If the user has set \texttt{ISTEP = 'Q'} at \texttt{PX6}, then \texttt{PITRANS} will “quit” its \texttt{PX6} activities and return to the program’s \texttt{PX3} control point to prompt the user for his next instruction(s).

If the user sets \texttt{ODATA = 'S' or 's'} at the program’s \texttt{PX3} control point—to ask for \textit{nodal stress} or \textit{strain results} from the solution database—\texttt{PITRANS} will go to its \texttt{PX8} control point, will display its current “load table” on the screen and save it in a formatted (text) output file called \textit{case.step} (if it hasn’t already done so), and prompt the user to specify (via the \texttt{STEP} input parameter) what he wants the program to do next:
If the user has set \texttt{STEP} = \texttt{P} or \texttt{S} at \texttt{PX8}, then \texttt{PITRANS} will print information from another part of the “load table,” then return to \texttt{PX8} to ask the user what he wants the program to do next.

If the user has set \texttt{STEP} = \texttt{N} at \texttt{PX8} (where \texttt{N} is a valid step number for the case at hand), then \texttt{PITRANS} will ask the user for more information, as discussed in the next three sub-paragraphs:

With a valid step number in hand, \texttt{PITRANS} will go to its \texttt{PX9} control point and prompt the user to specify a valid layer number—by setting the \texttt{LAYER} parameter equal to an integer value (to select a particular layer) or to zero (to select all layers) for some or all the elements in the model. \texttt{PITRANS} will check to ensure that \texttt{LAYER} is valid and will return to \texttt{PX9} if it is not.

Then, with a valid \texttt{LAYER} specification in hand, \texttt{PITRANS} will check the \texttt{nr840} and \texttt{nr880} parameters that it retrieves from \texttt{case.sav} to determine whether or not the user’s model has any sandwich or solid elements in it. If it does, \texttt{PITRANS} will go to its \texttt{PX9a} control point and prompt the user to specify (\textit{via} the \texttt{SLOC} input parameter) that he wants the information that he is requesting for the centroid (if \texttt{SLOC} = \texttt{C}), or for the integration points (if \texttt{SLOC} = \texttt{I}) or for the node points (if \texttt{SLOC} = \texttt{N}) of each of the elements for which that information is available. If the user does not respond to this prompt by setting \texttt{SLOC} equal to one of those three characters, \texttt{PITRANS} will return to \texttt{PX9a} and prompt him to specify \texttt{SLOC} again.

Then, with a valid \texttt{SLOC} specification in hand, \texttt{PITRANS} will examine (a) the \texttt{plasx} parameter that it retrieves from \texttt{case.sav} to determine whether or not plasticity effects are taken into account in the user’s model and (b) a logical-type variable called \texttt{makeplas} that \texttt{PITRANS} initializes by making it \texttt{.false.} and changes under the conditions that are described at the end of this sub-paragraph. If \texttt{PITRANS} finds that plasticity effects exist and that the user has never answered or has given a negative answer to the program’s prompt for him to specify whether or not he wants plastic stains output for a previously selected load step, \texttt{PITRANS} will pause at its \texttt{PX9b} control point and prompt the user to specify (\textit{via} the \texttt{IPLAST} input parameter) whether or not he wants \texttt{PITRANS} to give him plastic strains output for the current \texttt{STEP}: if the user responds to this prompt by setting \texttt{IPLAST} = \texttt{N}, \texttt{PITRANS} will set \texttt{makeplas} = \texttt{.false.} and will not extract plastic strains information from the solutions database; if the user responds by not setting \texttt{IPLAST} or by setting it to anything other than \texttt{N}, \texttt{PITRANS} will set \texttt{makeplas} = \texttt{.true.} and will extract plastic strains information from the solutions database.
In any event, PITRANS will then go to its PX9c control point, where it will prompt the user to specify (via the FRAME input parameter) that he wants PITRANS to generate and give him the information that he wants using each element’s “material” coordinates reference frame (by setting FRAME = ‘M’) or that he wants PITRANS to generate and give it to him using each element’s “fabrication” coordinates reference frame (by not setting FRAME or by setting it equal to anything other than ‘M’).

Then, with STEP, LAYER & FRAME in hand (and with SLOC & IPLAST also in hand, if they are needed), PITRANS will extract the nodal stress and strain results (and the plastic strains, if any) that the user requested from the STAGS solutions database and give him the nodal stresses in an output file called case.elmT (and the plastic strains in a second output file called case.elmP.N.LAYER, if he asked for them). With that done, PITRANS will return to its PX8 control point and prompt the user for his next instruction(s).

If the user has set ISTEP = ‘Q’ at PX8, then PITRANS will “quit” its PX8 activities and return to the program’s PX3 control point to prompt the user for his next instruction(s).

Last (but far from least), if the user responds to PITRANS’ PX3 prompt by setting ODATA equal to ‘Q’ or ‘q’, PITRANS will close all active files and terminate the current execution.

Example # 1

In this example, the analyst used PITRANS to generate a PATRAN-compatible displacements vector for STAGS’ tiny “pcats” test case, demonstrating that it was much easier for him to generate that output file using PITRANS in its “interview” operating mode than it was for him to read about how to do so. The following “listing” shows PITRANS’ output to his screen (in blue type) and his keyboard input into the program (in red type) when he executed the program via the OS-level pitrans command in a working directory that contained the pcats.sav model file and the pcats.res and pcats.rst solution files:

```
1: [working directory] > pitrans
4: Enter output type: (P)ATRAN, (I)DEAS = Patran
5: Enter Stags Case Name = pcats
6:
7: Data: (M)odel only data (nodes, elements, bc, etc)
8: (V)ector results (displ, veloc, accel, forces)
9: (I)ntegrated Stresses -or- Mid-Surface Strains
10: (S)tress and Strains (equivalent nodal values)
11: (Q)uit making STAGS plot files ............... = Vector
```
12: Data Type: (D)isplacement, (V)elocit, (A)cceleration, Reaction-(F)orces = Displ
13: Enter model type: (F)ull or (C)ondensed: Full
14:
15: the load/time step list has been stored in the file: pcats.step
16:
17: there are 12 loadsteps for this case
18: ===============================================================================
19: loadstep of initial plastic yield: 8
20:                                                                                                                                                                                                 
21: | Loadstep | Load Factor A  | Load Factor B  | time         | # of modes |
22: |          | 1.000000E-02  | 0.000000E+00  | 0.000000E+00 | 0          |
23: | 1        | 1.000000E-02  | 0.000000E+00  | 0.000000E+00 | 0          |
24: | 2        | 2.000000E-02  | 0.000000E+00  | 0.000000E+00 | 0          |
25: | 3        | 2.572181E-02  | 0.000000E+00  | 0.000000E+00 | 0          |
26: | 4        | 3.286878E-02  | 0.000000E+00  | 0.000000E+00 | 0          |
27: | 5        | 4.301969E-02  | 0.000000E+00  | 0.000000E+00 | 0          |
28: | 6        | 5.709739E-02  | 0.000000E+00  | 0.000000E+00 | 0          |
29: | 7        | 7.361708E-02  | 0.000000E+00  | 0.000000E+00 | 0          |
30: | 8        | 9.189031E-02  | 0.000000E+00  | 0.000000E+00 | 0          |
31: | 9        | 1.091103E-01  | 0.000000E+00  | 0.000000E+00 | 0          |
32: | 10       | 1.240358E-01  | 0.000000E+00  | 0.000000E+00 | 0          |
33: | 11       | 1.280000E-01  | 0.000000E+00  | 0.000000E+00 | 0          |
34:                                                                                                                                                                                                 
35: (P)revious Group, (Q)uit, (S)tart Over or Load-Step No.: 10
36: File: pcats.dis.10 successfully created.
37:                                                                                                                                                                                                 
38: there are 12 loadsteps for this case
39: ===============================================================================
40:                                                                                                                                                                                                 
41: | Loadstep | Load Factor A  | Load Factor B  | time         | # of modes |
42: |          | 1.000000E-02  | 0.000000E+00  | 0.000000E+00 | 0          |
43: | 1        | 1.000000E-02  | 0.000000E+00  | 0.000000E+00 | 0          |
44: | 2        | 2.000000E-02  | 0.000000E+00  | 0.000000E+00 | 0          |
45: | 3        | 2.572181E-02  | 0.000000E+00  | 0.000000E+00 | 0          |
46: | 4        | 3.286878E-02  | 0.000000E+00  | 0.000000E+00 | 0          |
47: | 5        | 4.301969E-02  | 0.000000E+00  | 0.000000E+00 | 0          |
48: | 6        | 5.709739E-02  | 0.000000E+00  | 0.000000E+00 | 0          |
49: | 7        | 7.361708E-02  | 0.000000E+00  | 0.000000E+00 | 0          |
50: | 8        | 9.189031E-02  | 0.000000E+00  | 0.000000E+00 | 0          |
51: | 9        | 1.091103E-01  | 0.000000E+00  | 0.000000E+00 | 0          |
52: | 10       | 1.240358E-01  | 0.000000E+00  | 0.000000E+00 | 0          |
53: | 11       | 1.280000E-01  | 0.000000E+00  | 0.000000E+00 | 0          |
54:                                                                                                                                                                                                 
55: (P)revious Group, (Q)uit, (S)tart Over or Load-Step No.: Q
56:                                                                                                                                                                                                 
57: Data: (M)odel only data (nodes, elements, bc, etc)
58: (V)ector results (displ, veloci, accel, forces)
59: (I)ntegrated Stresses -or- Mid-Surface Strains
60: (S)tress and Strains (equivalent nodal values)
61: (Q)uit making STAGS plot files ................. = Q
62: [working directory] >

Note that the user’s responses on lines 4, 11, 12 and 13 of this “interview session” are character strings that have more than a single character in them. PITRANS reads and prints the user’s
Appendix C

A complete answer to most of its prompts, but it only uses the first character (in most cases). The input descriptions on the preceding pages do not indicate where PITRANS does that, so the interested reader should rely on the more complete descriptions of the program’s input records in the “batch mode input requirements” part of this Appendix. The printouts on lines 15 and 36 of this listing informed our user that PITRANS generated two text output files during his execution of the program—the pcats.step file that contains the same information that the program printed to the screen on lines 17–33 of the listing (and again on lines 38–53), and the PATRAN-compatible pcats.dis.10 file that contains the displacement results that he requested for load step # 10.

Example # 2

In this example, the analyst used PITRANS to generate an IDEAS-compatible output file containing mid-surface strains and curvatures for the same (pcats) test case—demonstrating again how much easier it was for him to generate that output using PITRANS in its “interview” operating mode than it was for him to read about how to do so. The following “listing” shows PITRANS’ output to his screen (in blue type) and his keyboard input into the program (in Red type) when he executed the program via the OS-level pitrans command in a working directory that contained the pcats.sav model file and the pcats.res and pcats.rst solution files:

```
1: [working directory] > pitrans
4: Enter output type: (P)ATRAN, (I)DEAS = IDEAS
5: Enter Stags Case Name = pcats
6: Data: (M)odel only data (nodes, elements, bc, etc)
7: Vector results (displ, veloc, accel, forces)
8: Integrated Stresses or Mid-Surface Strains
9: Stress and Strains (equivalent nodal values)
10: (Q)uit making STAGS plot files ................ = Integrated S or M
11: Integrated Type: (S)treses = Resultants & Moments
12: (M)id-Surface Strains & Curvatures = M
13: 14: 15: the load/time step list has been stored in the file: pcats.step
16: 17: there are 12 loadsteps for this case
18: ==============================================================
19: loadstep of initial plastic yield: 8
20: Loadstep Load Factor A Load Factor B time # of modes
21: 0 1.000000E-02 0.000000E+00 0.000000E+00 0
22: 1 1.000000E-02 0.000000E+00 0.000000E+00 0
23: 2 1.000000E-02 0.000000E+00 0.000000E+00 0
24: 3 2.000000E-02 0.000000E+00 0.000000E+00 0
25: 4 3.286678E-02 0.000000E+00 0.000000E+00 0
```
Note that the user’s responses on lines 4 and 11 of this “interview session” are multi-character strings. For these responses, PITRANS reads and prints the full character string, extracts the first character from it, and uses that character to determine what the user wants and to do what it has to do next. Note that this is not the “case” on line 5 (pun intended) where the program assumes that the user’s case name is whatever he gives the program on that line—all of it. The printouts on lines 15 and 37 of this listing informed our user that PITRANS generated two output files during his execution of the program—the *pcats.step* file that contains the same information that the program printed to the screen on lines 17–33 of the listing (and again on lines 39–54), and the IDEAS-compatible *pcats.elmR.10* file that contains the mid-surface strain and curvature results that he requested for load step # 10.
Input records (for interview- and batch-mode operations)

On this and on the following eleven pages, we give “formal” descriptions of the twelve input records (user responses) that were referenced in the flow chart on page C-3 and in the Input requirements and output files discussions part of this Appendix—in the traditional input record format that is used throughout this and other STAGS user manuals. Some of these records are required whenever the user executes the PITRANS program, and others are only needed when the user asks the program to do certain things. All of that will be clear from the description of each of these records and from the “navigation” instructions that appear at the end of it.

PX-1 Output Format record

The first thing that PITRANS does when the user executes the program is to print a short program-version header followed by its initial (station PX1) prompt that asks the user to specify the format that he wants the program to use for all of the output files that it generates for him during the current execution (where PX1 is the first of the twelve prompt and control addresses in the flow chart on page C-3). The following PX-1 input record documents the answer—a character string called FORM that may contain up to but not more than 72 alphanumeric characters—that PITRANS expects the user to give it in his response to that prompt:

```
FORM
```

FORM output format (72 characters or less); see the note, below

**Note:** PITRANS reads and displays the complete FORM string, extracts the first character from it, and uses that character to set its internal control parameters to generate all of the output files that it produces in PATRAN-compatible formatted files (if the first character is ‘P’ or ‘p’) or in IDEAS-compatible formatted files (if the first character is ‘I’ or ‘i’). If the first character is anything other than one of those four possibilities, PITRANS will reissue its PX1 control point prompt and wait for the user’s next response to the same question. Then—after the user’s valid response has been read and processed—PITRANS will move forward to its next prompt station (and input record).

* go to PX-2 *
PX-2  Stags Case Name record

This record specifies the case name for the case at hand.

CASE

CASE    case name (an alphanumeric character string that identifies the case at hand); see the important Notes, below

Note 1: In PITRANS, the CASE character string is long enough to contain up to (but not more than) 72 characters. Some analysts who are using PITRANS with cases that have names that are much shorter than that might be tempted to respond to the program’s PX2 control station prompt with a short case name followed by one or more blank characters followed by comments or other things. The PITRANS program is not “smart” enough to realize that the user might give it a short case name (which cannot contain any blank characters) followed by other stuff, and it tries to use the entire character string instead of the short substring that the user wants it to use. Please do not let PITRANS do that.

Note 2: PITRANS will need the case.sav file that s1 generated for the user’s case to produce any output files for the case at hand—and it will need the case.res and case.rst files that s2 generated to produce any output files that contain displacements, velocities, reaction forces, stresses, strains and/or other results that are contained in those files or that must be generated with information that those files contain. The case.sav file must be present in the user’s working directory under any circumstances. The case.res and case.rst files must also be there when they are needed. PITRANS will read each of these files when and if it needs to do so—but will not change any of them under any circumstances.

Note 3: PITRANS will terminate its execution prematurely (stop) if it does not find the case.sav file that it expects (and needs) in the user’s working directory. If PITRANS finds and opens that file successfully, it will move on to its central PX3 control station, where it expects the user to respond with the following (PX-3) program-control input record.

go to PX-3
PX-3 Output Data Option record

This record is requested and read at PITRANS’ central control point—the PX3 control station in the flow chart on page C-3 to initiate one or another of the program’s data extraction, translation and output activities—or to quit the PITRANS program. This is where PITRANS displays its

Data: (M)odel only data (nodes, elements, bc, etc)
(V)ector results (displ, veloc, accel, forces)
(I)ntegrated Stresses -or- Mid-Surface Strains
(S)tress and Strains (equivalent nodal values)
(Q)uit making STAGS plot files ......... = >

prompt and waits for the user’s answer to the what do you want me to do next question that it poses—via the very first character in his specification of the 72-character-long OPTION answer string on this input record:

______________________________
OPTION
______________________________

OPTION(1:1)  primary program-control input parameter:

= ‘M’ or ‘m’ — generate a PATRAN- or an IDEAS-compatible output file containing information about the STAGS model

= ‘V’ or ‘v’ — generate one or more PATRAN- or IDEAS-compatible output files that contain displacement, velocity or reaction force results for the specified load step(s)

= ‘I’ or ‘i’ — generate one or more PATRAN- or IDEAS-compatible output files that contain integrated stress results for the specified load step(s)

= ‘S’ or ‘s’ — generate one or more PATRAN- or IDEAS-compatible output files that contain nodal stress/strain results for the specified load step(s)

= ‘Q’ or ‘q’ — stop generating output files!!!

if  ( OPTION(1:1) = ‘M’ OR ‘m’ ) then
  go to PX-7
elseif ( OPTION(1:1) = ‘V’ OR ‘v’ ) then
  go to PX-4
elseif ( OPTION(1:1) = ‘I’ OR ‘i’ ) then
  go to PX-5
elseif ( OPTION(1:1) = ‘S’ OR ‘s’ ) then
  go to PX-8
elseif ( OPTION(1:1) = ‘Q’ OR ‘q’ ) then
  finished!!!
else
  go to PX-3
endif
PX-4  Vector Type record

This record is read at PITRANS’ PX4 control station (in the flow chart on page C-3), which PITRANS reaches if and only if the user has specified that ODATA(1:1) = ‘V’ or ‘v’ on his most recent PX-3 record.

VTYPE

VTYPE(1:1)  vector-type control parameter:

= ‘D’ or ‘d’ – generate one or more PATRAN- or IDEAS-compatible output files that contain displacement results for the specified load step(s)

= ‘V’ or ‘v’ – generate one or more PATRAN- or IDEAS-compatible output files that contain velocity results for the specified load step(s)

= ‘F’ or ‘f’ – generate one or more PATRAN- or IDEAS-compatible output files that contain reaction forces for the specified load step(s)

if  ( VTYPE(1:1) = ‘D’ or ‘d’ )  then
  go to PX-7
elseif ( VTYPE(1:1) = ‘V’ or ‘v’ )  then
  go to PX-7
elseif ( VTYPE(1:1) = ‘F’ or ‘f’ )  then
  go to PX-7
else
  go to PX-4
endif
**PX-5 Integrated Stress Type record**

This record is read at PITRANS' PX5 control station, which PITRANS reaches if and only if the user has specified that ODATA(1:1) = ‘I’ or ‘i’ on his most recent PX-3 record.

---

**STYPE**

STYPE(1:1) integrated-stress-type control parameter:

- = ‘S’ or ‘s’ – generate one or more PATRAN- or IDEAS-compatible output files that contain integrated stresses in each element for the specified load step(s)
- = ‘M’ or ‘m’ – generate one or more PATRAN- or IDEAS-compatible output files that contain mid-surface strains in each element for the specified load step(s)

```plaintext
if ( STYPE(1:1) = ‘S’ OR ‘M’ OR ‘s’ OR ‘m’ ) then
  go to PX-6
else
  go to PX-5
endif
```
PX-6 Next Step record

This record is read at PITRANS' PX6 control station, which PITRANS reaches if and only if the user has specified that ODATA(1:1) = ‘I’ or ‘i’ on his most recent PX-3 record and if the user has specified that STYPE(1:1) = ‘S’ or ‘M’ (or ‘s’ or ‘m’) on his most recent PX-5 record. This is where the user selects a valid step number from the “load table” for the analysis (or analyses) that produced the case.res and case.rst files in his working directory. This is a little bit tricky because the ISTEP input parameter can contain an integer step number (N, say) or a character string (A, say)—the first of these being the case where the user wants to specify a valid load step for which he wants output of the type that he requested on his PX-5 record, and the second of these being the case when he wants to direct PITRANS to do something.

ISTEP = N or A

<table>
<thead>
<tr>
<th>ISTEP</th>
<th>next step number or action string:</th>
</tr>
</thead>
<tbody>
<tr>
<td>= N</td>
<td>generate a PATRAN- or an IDEAS-compatible output file that contains integrated stresses or mid-surface strains in each element for load step # N</td>
</tr>
<tr>
<td>= A(1:1) = ‘P’</td>
<td>display an earlier section of the “load table” than is currently being displayed on PITRANS’ output screen</td>
</tr>
<tr>
<td>= A(1:1) = ‘S’</td>
<td>display the very first section of the “load table” instead of the section that is currently being displayed</td>
</tr>
<tr>
<td>= A(1:1) = ‘Q’</td>
<td>quit generating integrated stress or mid-surface strains output files; return to PX3 to prompt and wait for further instructions,</td>
</tr>
</tbody>
</table>

if (ISTEP = N) then

go to PX-6

elseif (ISTEP = S(1:1) = ‘P’ or ‘S’) then

go to PX-6

elseif (ISTEP = S(1:1) = ‘Q’) then

go to PX-3

else

go to PX-6
endif
PX-7 Model Type record

This record is read at PITRANS’ PX7 control station, which PITRANS reaches if and only if the user has specified that ODATA(1:1) = ‘M’ or ‘m’ on his most recent PX-3 record—or if he has specified that ODATA(1:1) = ‘V’ or ‘v’ there and that he has specified that VTYPE(1:1) = ‘D’ or ‘V’ or ‘F’ (or ‘d’ or ‘v’ or ‘f’) on his most recent PX-4 record. At PX7, PITRANS prompts the user to specify (via the MTYPE input parameter) that he wants PITRANS to do any calculations that it needs to do using his STAGS model in the program’s “full model” computational mode or in the program’s “condensed model” mode when it calculates forces, stresses and other results. When the user specifies that he wants the “full model” option, PITRANS will perform its computations keeping nodes that occupy the same locations as other nodes in-place as “doubled” nodes. When the user specifies that he wants the “condensed model” option, PITRANS will coalesce each “doubled” node into a single “master” node. This is important when downstream processors cannot handle “doubled” node situations properly. STAGS and its utility programs can handle those situations correctly, so it is rarely (if ever) necessary for STAGS users to choose the “condensed model” option.

PITRANS expects to read the following (Model Type) input record in response to this PX7 prompt:

```
MTYPE

MTYPE(1:1) model type parameter:
    = ‘F’ – perform force, stress and other calculations using the program’s “full model” computation mode (the usual situation); or
    = ‘C’ – perform force, stress and other calculations using the program’s “condensed model” mode
```

if ( MTYPE(1:1) = ‘F’ or ‘C’ or ‘f’ or ‘c’ ) then
    if ( ODATA(1:1) = ‘M’ or ‘m’ ) go to PX-3
    if ( ODATA(1:1) = ‘V’ or ‘v’ ) go to PX-8
else
    go to PX-7
endif
**PX-8 Next Step record**

This record is read at **PITRANS’ PX8** control station, which **PITRANS** reaches if and only if the user has specified that

\[
\text{ODATA}(1:1) = \text{‘S’ or ‘s’} \quad \text{on his most recent PX-3 record, or that}
\]

\[
\text{ODATA}(1:1) = \text{‘V’ or ‘v’} \quad \text{at PX3, and that VTYPE(1:1) = ‘D’ or ‘V’ or ‘F’ (or ‘d’ or ‘v’ or ‘f’) at PX4, and that MTYPE(1:1) = ‘F’ or ‘C’ (or ‘f’ or ‘c’) at PX7}
\]

Under these circumstances, this is where the user selects a valid step number from the “load table” for the analysis (or analyses) that produced the **case.res** and **case.rst** files in his working directory. The **STEP** input parameter can contain an integer step number (N, say) or a character string (A, say)—the first of these being the case where the user wants to specify a valid load step for which he wants output of the requested type, and the second of these being the case when he wants to direct **PITRANS** to do something.

**STEP = N or A**

<table>
<thead>
<tr>
<th>STEP</th>
<th>next step number or action string:</th>
</tr>
</thead>
<tbody>
<tr>
<td>= N</td>
<td>– generate a PATRAN- or an IDEAS-compatible output file that contains the requested information for load step # N</td>
</tr>
<tr>
<td>= A(1:1) = ‘P’</td>
<td>display an earlier section of the “load table” than is currently being displayed on <strong>PITRANS’</strong> output screen</td>
</tr>
<tr>
<td>= A(1:1) = ‘S’</td>
<td>display the first section of the “load table” instead of the section that is currently being displayed</td>
</tr>
<tr>
<td>= A(1:1) = ‘Q’</td>
<td>quit generating output files of the requested type</td>
</tr>
</tbody>
</table>

```javascript
if (STEP = N) then
    if (ODATA(1:1) = ‘V’ or ‘v’) go to PX-8
    if (ODATA(1:1) = ‘S’ or ‘s’) go to PX-9
elseif (STEP = A(1:1) = ‘P’ or ‘S’) then
    go to PX-8
elseif (STEP = A(1:1) = ‘Q’) then
    go to PX-3
else
    go to PX-8
endif
```
PX-9 Layer record

This record is read at PITRANS’ PX9 control station, which PITRANS reaches if and only if the user has specified that ODATA(1:1) = ‘S’ or ‘s’ on his most recent PX-3 input record (at PX3) and that he has selected a valid step number in his most recent PX-8 input record (at PX8). This is where PITRANS prompts the user to specify the layer number (in the LAYER input parameter) for which he wants the program to give him the output that he has requested for each laminated element in the model that has a layer with that number—in which case a valid LAYER number will be in the range \(1 \leq \text{LAYER} \leq \text{maxlayer}\), where maxlayer is the highest layer number in the model (a parameter that PITRANS determines by examining all of the elements in the model)— or that he wants the program to give him the output that he has requested at layer number “0” (i.e., for the top and bottom laminates) in each element:

```
if ( LAYER is a valid layer number ) then
    if ( nr840 > 0 or nr880 > 0 ) then
        go to PX-9a
    elseif ( plax and .not. makeplax ) then
        go to PX-9b
    else
        go to PX-9c
    endif
else
    go to PX-9
endif
```
PX-9a Output Point Location record

This record is read at PITRANS’ PX9a control station, which PITRANS reaches if and only if the user has specified a valid LAYER number in the preceding PX-9 record at the program’s PX9 control point AND if the user’s model contains one or more sandwich or solid elements in it. This is where PITRANS prompts the user to specify the location(s) at which he wants it to generate the output that he has requested in each sandwich or solid element for which LAYER is a valid layer number (when LAYER > 0) or for the top and bottom laminates (when LAYER = 0)—in the single-character SLOC input parameter that the program expects the user to specify here:

SLOC

<table>
<thead>
<tr>
<th>SLOC</th>
<th>location(s) where output is wanted:</th>
</tr>
</thead>
<tbody>
<tr>
<td>= ‘C’</td>
<td>generate output at the centroid of the selected layer(s)</td>
</tr>
<tr>
<td>= ‘I’</td>
<td>generate output at each integration point of the selected layer(s)</td>
</tr>
<tr>
<td>= ‘N’</td>
<td>generate output at each node point of the selected layer(s)</td>
</tr>
</tbody>
</table>

if (SLOC = ‘C’ or ‘I’ or ‘N’ or ‘c’ or ‘i’ or ‘n’) then
  if (plasx and .not. makeplas) then
    go to PX-9b
  else
    go to PX-9c
  endif
else
  go to PX-9a
endif
**PX-9b Plastic Strains record**

This record is read at PITRANS’ PX9b control station, which PITRANS reaches if and only if the user has specified a valid output location (SLOC) number in the preceding PX-9a record at the program’s PX9a control point AND if the user’s model takes plasticity into account AND if PITRANS’ internal logical variable `makeplas` is .false. when the user reaches this control point—where PITRANS asks the user the question “Do you want to examine plastic strains? (y or n)” and waits for him to respond (via the single-character IPLAST input parameter) on the PX-9b input record that is described here:

---

**IPLAST**

| IPLAST | do I want plastic strains output???
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>= ‘N’ or ‘n’ – No, I do not: do not give me a plastic strains output file for this loading step</td>
</tr>
<tr>
<td></td>
<td>≠ ‘N’ or ‘n’ – Yes, I do: please give me a plastic strains output file for this and for all subsequent load steps; and do not ask me this question again</td>
</tr>
</tbody>
</table>

**Note:** PITRANS initializes `makeplas` to be .false. at the outset of each execution of the program—and will refrain from extracting plastic strains from the user’s case.res/case.rst database (and from giving the user output files containing that information) until the user changes `makeplas` to be .true. by giving PITRANS anything other than an ‘N’ or ‘n’ response to this PX9b prompt.

---

*go to PX-9c*
PX-9c Reference Frame record

This record is read at PITRANS’ PX9c control station, which the program reaches if and only if the user has selected a valid LAYER number at PX9 (and has specified the SLOC and/or IPLAST parameters successfully at PX9a and PX9b, as and if necessary). This is where PITRANS asks the user to specify the (fabrication or material) reference coordinate frame that he wants the program to use for all of the stress and strain computations has to do in each element, via the single-character FRAME parameter on the following input record:

<table>
<thead>
<tr>
<th>FRAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference coordinate frame:</td>
</tr>
<tr>
<td><code>M</code> or <code>m</code> – use the layer’s (or element’s) material coordinates frame</td>
</tr>
<tr>
<td>≠ <code>M</code> or <code>m</code> – use the layer’s (or element’s) fabrication coordinates frame</td>
</tr>
</tbody>
</table>

Note: PITRANS sets FRAME = ‘F’ if the user does not set it equal to ‘M’ or ‘m’.

go to PX-8
Example #3

In this example, the analyst decided that he wanted resultants and moments (instead of mid-surface strains and curvatures) for his pcats test case—at load steps 9, 10 and 11. Not wanting to plow through another of PITRANS’ painful interview sessions—and remembering what he did in the second example (above)—he constructed the following pcats.pix “script” file that answers PITRANS’ prompts in the way that he needs to do so to get the output that he wants:

```
1: I $ PX-1 IDEAS-compatible output desired
2: pcats
3: Integrated S or M $ PX-3 Output Data OPTION
4: R $ PX-5 Resultants and moments
5: 9 $ PX-6 ISTEP = 9
6: 10 $ PX-6 ISTEP = 10
7: 11 $ PX-6 ISTEP = 11
8: Q $ PX-6 Quit specifying steps
9: Q $ PX-3 Quit PITRANS !!!
```

The **red** items in this text file are our analyst’s answers to the prompts that he expects PITRANS to make, and the **black** items are in-line comments that he expects the program to ignore. Note that he has been careful not to put anything other than his *case* name on the second line of this input script.

Then, with *pcats.sav, pcats.res, pcats.rst* and *pcats.pix* in his working directory, our analyst executed PITRANS with the following OS-level command:

```
% pitrans < pcats.pix
```

When PITRANS was finished with this job, our happy analyst found the following output files in his working directory:

- **pcats.step** load step table
- **pcats.elmI.9** IDEAS-compatible resultant & moment results for step # 9
- **pcats.elmI.10** IDEAS-compatible resultant & moment results for step # 10
- **pcats.elmI.11** IDEAS-compatible resultant & moment results for step # 11
- **pcats.sot.0** PITRANS “journal” file for this run
STAGS Shell Surface Differential Geometry

Definition of Basic Differential Geometry Terms

Slope information to be used to generate higher-order shell imperfections can be computed from the relationships defining the shell surface. The basic relationships, given in (4.1) on page 4-8, are repeated here:

\[ x = x(X, Y) \quad y = y(X, Y) \quad z = z(X, Y) \]  \hspace{1cm} (K.1)

where \((x, y, z)\) are branch coordinates of the position vector \(r\) of a point defined by functions of the \((X, Y)\) surface coordinates (see Section 4.2 “Shell Unit” on page 4-8). Such functions are defined for each of the standard shell surfaces in “Standard Shell Surfaces” on page 6-2. Slope information required by user-written subroutine DIMP can be expressed as a function of the bending numbers \(\beta_x, \beta_y, \text{ and } \gamma\), which in turn depend on the knowledge of the Lamé coefficients (the First Fundamental Form) of the shell. It is possible to compute the Lamé coefficients by differentiation of the functional form provided in “Standard Shell Surfaces” for each shell unit. Without delving into shell theory, we present here a short summary of the required information for each STAGS shell type. Only the leading terms are presented for LAME and the elliptic cylinder (\(\text{ISHELL} = 9\)). In most situations, the leading terms involving derivatives of the displacement normal to the surface are sufficient to define the higher-order imperfection field to better precision than can be measured. The user should note that the imperfection field for the coordinates (the translations) is always expressed in units of distance (as opposed to angles or other generalized curvilinear coordinates). One

* See, for example:
  
  Kraus, H., “Thin Elastic Shells,” John Wiley & Sons, Inc., New York, 1967; and/or
  
  Dym, C.L., “Introduction to the Theory of Shells (revised printing),” Hemisphere Publishing Corp., New York, 1990; and/or
  
  
should also note that although for some shell units formulas for the drilling freedom $\gamma$ are included, this is a small quantity that could without harm be set to zero.

Note that in what follows, the perturbed coordinates are labeled $(u, v, w)$, and are always expressed in the $(X', Y', Z')$ shell coordinate system. The most important coordinate for thin shell imperfections is $w$, which points in the direction of the shell normal.

### Bending Numbers for Arbitrary Shell

Omitting details that can be found in most texts on shell theory or differential geometry, we define the in-plane bending numbers $\beta_i$ by the following relationship:

$$
\zeta_j = w_{ij} + \sum_k \frac{n \cdot r_{kj} u_k}{\sqrt{r_{ik} \cdot r_{ik}}} \\
\beta_j = \frac{\zeta_j}{\sqrt{r_{ij} \cdot r_{ij}}}
$$

(K.2)

The comma notation indicates differentiation; for example, $r_{ij} = \frac{\partial r}{\partial t}$.

$n$ is a unit vector pointing in the direction of the outward normal to the shell, and $u_k$ are the components of the in-plane displacements expressed in a system tangent to the coordinate lines. This equation applies to arbitrary tangent lines, including those that are not orthogonal. The user will discover in practice that only the term involving the normal coordinate $w$ will contribute to the imperfection (in fact, it is very difficult to measure anything else). The drilling freedom $\gamma$ can be ignored.

For the shell types orthogonalized by STAGS, the user must express his imperfection displacements and rotations in the orthogonal system. However, the imperfection displacements are still expressed as functions of the original $(X, Y)$ surface coordinates. To express the new direction in terms of the original $(X, Y)$, we make a change of variables that orthogonalizes the system, and apply that to (K.2). The final result is
Again, note that only the terms involving the undeformed surface and the normal coordinate are important. This fact will be reflected in the next section.

**ISHELL = 1 (LAME) Option**

This option permits the user to define his own shell type, with the possibility that the tangent coordinate line do not form an orthogonal set. The Lamé coefficients \( \alpha^2 \) for an orthogonal system are defined by the following relationships:

\[
\beta_1 = \frac{\hat{n} \cdot [r_{11} u_1 / \alpha_1 + (r_{12} - c \cdot r_{11}) u_2 / \alpha_2] + w_1}{\alpha_1}
\]

\[
\beta_2 = (\xi + \eta + \zeta) / \alpha_2
\]

\[
\xi = w_{22} - c w_{11}
\]

\[
\eta = \hat{n} \cdot [r_{12} - c r_{11}] u_1 / \alpha_1
\]

\[
\zeta = \hat{n} \cdot [r_{11} c^2 - 2 r_{12} c + r_{22}] u_2 / \alpha_2
\]

\[
c = \frac{r_{11} \cdot r_{22}}{r_{11} \cdot r_{11}}
\]

\[
\alpha_1 = \sqrt{r_{11} \cdot r_{11}}
\]

\[
\alpha_2 = \sqrt{r_{22} \cdot r_{22} - c (r_{11} \cdot r_{22})}
\]

(K.3)

Note that all these quantities are easily computed given (K.1). The bending numbers for LAME are

\[
\alpha_i = \sqrt{g_{ii}} = \sqrt{r_{11} \cdot r_{11}}
\]

\[
\alpha_j = \frac{r_{11} \cdot r_{11} - (r_{11} \cdot r_{11})^2}{r_{11} \cdot r_{11}}
\]

(K.5)
where the value of \( i \) and \( j \) range from 1 to 2, and where no summation is implied by repeated indices.

**Flat Plates (ISHELL = 2 or 3)**

Please see “RECTANGULAR PLATE” and “QUADRILATERAL PLATE” on page 6-4 for the defining equations. The bending numbers are

\[
\beta_i = \frac{w_i}{\alpha_i} \\
\beta_j = \frac{w_j}{\alpha_j} \\
\gamma = 0 
\]  
(K.6)

\[
\varphi_{ij} = w_{ij} - w_i \left( \frac{r_{ij} - r_i}{\alpha_i} \right) 
\]

and (this is almost never necessary)

\[
\gamma = \frac{1}{2} (v_{11} - u_{12}) 
\]  
(K.8)

**Annular Plate (ISHELL = 4)**

Please refer to “ANNULAR PLATE” on page 6-5 for the notation and defining equations. The bending numbers are

\[
\beta_1 = \frac{w_1}{X} \\
\beta_2 = \frac{w_{22}}{X} \\
\gamma = \frac{1}{2} \left( v_{11} - \frac{u_{22} - v}{X} \right) 
\]  
(K.9)

where \( X \) is the radial position coordinate within the annular plate segment.
Cylinder (ISHELL = 5)

Please refer to “CYLINDER” on page 6-5 for the notation and defining equations. The bending numbers are

\[
\begin{align*}
\beta_1 &= w_1 \\
\beta_2 &= \frac{w_2 - v}{R} \\
\gamma &= \frac{1}{2}(v_1 - \frac{u_2}{R})
\end{align*}
\]  
(K.10)

Cone (ISHELL = 6)

Please refer to “CONE” on page 6-6 for the notation and defining equations. The bending numbers are

\[
\begin{align*}
\beta_1 &= \frac{w_1}{\alpha_1} \\
\beta_2 &= \frac{w_2 - v/\alpha_1}{\alpha_2} \\
\gamma &= \frac{1}{2}\left(v_1 - \frac{u_2 + \frac{v\tan \phi}{\alpha_1 \alpha_2}}{\alpha_1 \alpha_2}\right)
\end{align*}
\]  
(K.11)

where the Lamé coefficients are defined by

\[
\begin{align*}
\alpha_1 &= \frac{1}{\cos \phi} \\
\alpha_2 &= r
\end{align*}
\]  
(K.12)

Sphere (ISHELL = 7)

Please refer to “SPHERE” on page 6-6 for the notation and defining equations. The bending numbers are
Torus (ISHELL = 8)

Please refer to “TORUS” on page 6-7 for the notation and defining equations. The bending numbers are

\[
\beta_1 = \frac{w_{r,1} - u}{\alpha_1} \\
\beta_2 = \frac{w_{r,2} - v \sin X}{\alpha_2} \\
\gamma = \frac{1}{2} \left( \frac{v_{z,1}}{\alpha_1} - \frac{u_{z,2} - v \cos X}{\alpha_2} \right)
\]

where the Lamé coefficients are defined by

\[
\alpha_1 = R_b \\
\alpha_2 = R_a + R_b \sin X
\]

Elliptic Cylinder (ISHELL = 9)

This is a special case where the shell coordinates are orthogonalized, as discussed above. Approximate formulas for the in-plane bending numbers will be given, and the drilling freedom \( \gamma \) will be ignored. Please refer to “ELLIPTIC CONE/CYLINDER” on page 6-8 for the notation and defining equations. The bending numbers are

\[
\beta_1 = \frac{w_{r,1} - c w_{r,2}}{\alpha_1} \\
\beta_2 = \frac{w_{r,2}}{\alpha_2} \\
\gamma = 0
\]

with constants in (K.16) defined as follows:
Paraboloid (ISHELL = 10)

Please refer to “PARABOLOID” on page 6-9 for the notation and defining equations. The bending numbers are

\[
\beta_1 = \frac{w_{x_1}}{\alpha_1} - u \frac{4R_a^2}{(\alpha_1 \alpha_2)^3} \\
\beta_2 = \frac{w_{x_2} - v/\alpha_1}{\alpha_2} \\
\gamma = \frac{1}{2} \left( \frac{v_{x_1}}{\alpha_1} - \frac{u_{x_2} - v/\left[\alpha_1 \alpha_2\right]}{\alpha_2} \right) \tag{K.18}
\]

where the Lamé coefficients are defined by

\[
\alpha_1 = \sqrt{1 + \eta^2} \\
\alpha_2 = \tilde{R}(x) \\
\eta = \frac{R_a}{X + R_b} \tag{K.19}
\]

Ellipsoid (ISHELL = 11)

Please refer to “ELLIPSOID” on page 6-10 for the notation and defining equations. The bending numbers are

\[
g_{22} = X_{ap}^2 (\tan \phi)^2 (\xi^2 \cos^2 Y + \sin^2 Y) \\
g_{12} = \sin Y \cos Y (\tan \phi)^2 X_{ap} (\xi^2 - 1) \\
g_{11} = 1 + (\tan \phi)^2 (\xi^2 \sin^2 Y + \cos^2 Y) \\
c = g_{12}/g_{22} \\
\alpha_1 = \sqrt{g_{11} - cg_{12}} \\
\alpha_2 = \sqrt{g_{22}}
\]
where the Lamé coefficients are defined by

\[ \alpha_1 = \sqrt{R_x^2 (\sin X_c)^2 + R_y^2 (\cos X_c)^2} \]
\[ \alpha_2 = R_{yz} \sin X_c \] (K.21)

**Hyperboloid** \((\text{ISHELL} = 12)\)

Please refer to “HYPERBOLOID” on page 6-11 for the notation and defining equations. The bending numbers are

\[ \beta_1 = \frac{w_{s1} - u R_x R_{yz} / \alpha_1^2}{\alpha_1} \]
\[ \beta_2 = \frac{w_{s2} - v}{\alpha_2} \frac{R_x}{\alpha_1 R_{yz}} \]
\[ \gamma = \frac{1}{2} \left( \frac{u_{s1} - v R_{yz} \cos X_c / \alpha_1}{\alpha_2} \right) \] (K.22)

where the Lamé coefficients are defined by

\[ \alpha_1 = \sqrt{1 + \eta^2} \]
\[ \alpha_2 = R(x) \]
\[ \eta = \frac{R_y^2 (x - x_1 + R_c)}{R_h^2 R(x)} \] (K.23)
Design Parameter Derivatives

Displacement gradients

The objective is to determine how the system unknowns change as a function of changes in model parameters that govern system behavior. The starting point for such an analysis is the governing equation that determines the system state:

\[ f(u, \lambda) = 0 \]  \hspace{1cm} (L.1)

where \( f \) is the nodal residual vector (internal minus external forces), and \( u \) are the nodal displacements and the system unknowns to be determined by application of (L.1). Equation (L.1) is a statement of Newton’s second law in a very general sense; such a system is usually nonlinear in its unknowns, and may require sophisticated solution techniques and considerable resources to solve. The system has a number of independent parameters that determine response, such as nodal positions and element topology, shell wall fabrication, and material data. In principle, any of these parameters can change as a design evolves. Our objective is to determine how the system unknowns change when such design parameters \( \lambda \) change. We assume at the outset that we have a solution at a given reference state (our current design and environment), and that we wish to see how these unknowns evolve for small changes in our current system definition. We obtain the governing equations by differentiating (L.1) to yield

\[ \frac{\partial f(u, \lambda)}{\partial u} u_{\lambda} + \frac{\partial f(u, \lambda)}{\partial \lambda} = 0 \]  \hspace{1cm} (L.2)

where \( u_{\lambda} \) is the displacement gradient vector field that we wish to compute, and where a subscripted comma followed by a variable denotes a derivative with respect to that variable. The first partial derivative is shorthand for the derivative of the residual with respect to each unknown in turn, with all others held constant. The second term is the partial derivative of the residual with respect to the design variable with all other unknowns and parameters held constant; this derivative is almost always local to each element, \( i.e. \), the total derivative is a sum of a contribution from each element taken independent of all the others. \( K \) is the stiffness matrix that is readily identified with the first partial derivative in (L.2); it is already available in FEM.
software. The second equation in (L.2) consists of a factoring of the stiffness matrix followed by
one or more “solve” operations with right hand sides $-f_{\lambda}$. Thus, the task is divided neatly into
two independent parts:

- Computation of local partial derivatives of the residual and assembly into $f_{\lambda}$.
- The solution of a system of linear equations with the same structure as the normal
  “systems matrix” in an ordinary FEM analysis.

The first task can be very complex—especially if element shape, topology, and nodal locations
are involved. For the time being, we shall concentrate on a very important subset of design
variables that yields a simplified analysis. We shall admit for variation all components of a wall
fabrication, such as layer thickness, material orientation angles and material properties. Any
number of such variations can be combined so that a very large class of design gradients can be
computed. It is fortuitous that no matter how complex a fabrication is in a shell analysis, it all
comes down in the end to the $ABD$ matrix that relates reference surface strains and curvatures
to force resultants and moments. Furthermore, the residual is a linear function of the $ABD$
matrix for the class of problems we are interested in; in what follows, we shall call this matrix
$C$ with components $ij$. $C$ is symmetric and positive definite, and has either 21 or 24 independent
components depending on the type of element used (the last three are transverse shear terms).
Since the entire response of the system to changes in $\lambda$ comes from $C$, we can express $f_{\lambda}$ as
follows by using the chain rule:

$$f_{\lambda} = \frac{\partial f}{\partial \lambda} = \sum_{ij} \frac{\partial f}{\partial C_{ij}} \frac{\partial C_{ij}}{\partial \lambda}$$

where we remove the bold face for components of $C$ to emphasize that they are treated one-by-
one. One can see readily that $f_{\lambda}$ is a linear combination of partials with respect to each individual
member of $C$. In addition, since the FEM analysis is a strictly linear function of $C$, one can compute
these partials by setting all the $C$ matrices in the system to zero, and replacing each component in turn by unity for fabrications affected by the design parameter. One must be
careful to preserve the symmetry of $C$: unity in the $ij$ slot must be matched by unity in the $ji$ slot.
Subsequently, one has to call a simple internal force calculation to obtain each of the 21 or 24
partials in (L.3). Clearly, if (L.3) holds, then

$$u_{\lambda} = \sum_{ij} u_{ij} \frac{\partial C_{ij}}{\partial \lambda}$$

$$Ku_{ij} = -\frac{\partial f}{\partial C_{ij}}$$

(L.4)
must also be true. The advantage of solving for the independent $u_{ij}$ vectors is that this particular operation needs to be done only once, no matter what contributes to changes in $C$. In fact, the second equation in (L.4) is the only part that has to be done with the full FEM machinery; the sum in the first of (L.4) can be performed in a post-processing environment. This is how it is done in STAGS. The user selects the fabrications to vary, computes the 21 or 24 gradient basis vectors $u_{ij}$, and stores them away. In a stapl or a star application, these vectors are multiplied by the coefficients of variation of $C$ with respect to a particular design variable to yield the final result. The variation coefficients of $C$ can be computed with simple software independent of STAGS. We have provided an example of such software, called GradC, which is described below. This GradC program allows linked variation with respect to material properties, layup thickness and orientation angles. The post-processing operation can be repeated for the variation of any different design parameter that affects the same $C$—and this can include a group of $C$’s that form multiple fabrications. In that case, there will be a set of basis vectors for each different fabrication. The user can obtain additional flexibility by assigning more fabrications to the model, even though they may have the same initial composition.

Strain gradients

Two ingredients are required to compute strain gradients as a function of some design variable $\lambda$. First, we must have the displacement gradients computed from the governing equations (L.2); this subject was covered in full in the previous section. The second requirement for the strain variation is the partial derivative or variation of the strain as a function of nodal displacements. For a linear analysis, this is easy to obtain, since one only has to turn off the nonlinearity in the strain-displacement relationship, and compute strain as a function of $u_\lambda$ instead of $u$. This is the same requirement as that expressed by the formula

$$B_\epsilon(X)u = \varepsilon$$  \hspace{1cm} (L.5)

where we have emphasized that $B$ depends only on the undeformed geometry $X$. For nonlinear analysis, we would require the relationship

$$B_{NL}(X, u)u = \varepsilon$$  \hspace{1cm} (L.6)

Unfortunately, this expression is not equivalent to substituting the displacement gradient for the displacements in the existing strain routines because $B$ itself is a function of the actual displacements of the problem. This is especially true of corotation software that is applied well before strains are computed.
Fortunately, there is an entirely different approach that can be used. We note that the internal force routines compute the first variation of the energy via

\[ \delta \mathbf{u}^T \mathbf{f}_{i,n} = \int (\sigma \cdot \delta \varepsilon) \delta A \]

\[ = \delta \mathbf{u}^T \left( \sum \tilde{w}_l \mathbf{B}_{Nl}^T \sigma_l \right) \]  \hspace{1cm} (L.7)

where we have arranged the stresses into a “vector” defined by a particular element. A corresponding vector exists for the strains, so that when “engineering” strains are used, the energy is the inner product of the stress and the strain. The \( \tilde{w} \) above are integration weights. If we examine (L.7) closely, we see that the integral of the gradient we desire is already in place, so long as we replace the stress vector with a one in the desired slot, and zero elsewhere. For example, in the E410 arrangement of strains, we have the ordering \( N_x, N_y, N_{xy}, M_x, M_y, M_{xy} \) for the stresses, and \( \varepsilon_x^0, \varepsilon_y^0, \varepsilon_{xy}^0, \kappa_x^0, \kappa_y^0, \kappa_{xy}^0 \) are the corresponding strains. If we wish to obtain \( \varepsilon_{xy}^0 \), then our “stress” vector would be zero except for a one for component \( N_{xy} \) for each integration point \( l \).

The resulting “internal force” vector \( \mathbf{i}_k \) provides the integral of the gradient of strain over the element as a function of a unit variation of a given displacement component. This relationship holds even in the presence of corotation and projection; in fact, results are improved when both are applied. The output of the internal force calculation is of course a nodal vector of the same length and structure as the internal force; we can call this vector \( \mathbf{i}_k \) the strain gradient coefficient vector. We summarize these operations by the following:

\[ \frac{(k)}{\sigma} = \mathbf{e}_k \]

\[ \delta \mathbf{u}^T \mathbf{i}_k = \int \frac{(k)}{\sigma} \delta \varepsilon dA \]

\[ = \delta \mathbf{u}^T \left( \sum \tilde{w}_l \mathbf{B}_{Nl}^T \frac{(k)}{\sigma} \right) \]  \hspace{1cm} (L.8)

where \( \mathbf{e}_k \) is an array with the same structure as stress, but with unity in position \( k \) and zero elsewhere. Now we see the significance of the subscript \( k \) on the force vector: it is the base variation of strain component \( k \) with respect to the displacement field. The integral of the strain variation with respect to a particular displacement gradient becomes

\[ \int \frac{e_{ki, i}}{\lambda} dA = \left( \frac{\delta \mathbf{u}}{\delta \lambda} \right)^T \mathbf{i}_k = \mathbf{i}_k^T \mathbf{u}_{\lambda} \]  \hspace{1cm} (L.9)

where the total derivative can be substituted for the ratio of variations since these variations in \( \mathbf{u} \) are restricted to those satisfying the equilibrium equation (L.1). The average (centroidal) gradient for the element is
where $A$ is the area of the element. These calculations can be done with existing software with a minimum of effort. For values at the integration points, one can set only that component corresponding to point $l$ to unity. Then the divisor in (L.10) becomes the integration weight $w_l$ instead of $A$, and the operations above are repeated for each integration point.

**Strain gradients with corotation**

The displacement gradient above refers to the deformational displacements in the local element frame. The inner product (L.10) is valid only when both $t_\beta$ and $u_\lambda$ are in the same coordinate frame. However, the displacement gradient available from STAGS is in the usual computational system. They first must be transformed to the element frame as a variation in total displacements, and then converted by the corotational projector into the local deformational system:

$$ u_\lambda = P u_\lambda^E $$  \hspace{1cm} (L.11)

where $P$ is the corotational projector, and $u_\lambda^E$ are the displacements taken from the STAGS archive and transformed into the element frame. This transforms the product in (L.7) to

$$ e_{k, \lambda} = \frac{f^T P u_\lambda^E}{A} $$  \hspace{1cm} (L.12)

This is the final result needed for display of the strain gradients in the element frame. The usual transformations apply for display of strain gradients in shell/fabrication coordinate frames.

**Gradient computation in STAGS**

Let us now summarize how STAGS accomplishes our objective to determine the gradient of the system displacement unknowns $u$. First, we must have a solution at the desired load and boundary environment for the reference design. Most often, such a solution is accomplished by solving (L.1) at a series of increasing load steps using all the solution tools available in STAGS. If the user desires to follow with a gradient analysis, he will specify this in the solution *.bin input file. One can request that the solution will stop at a given set of load steps, and instead of an eigenanalysis, perform a gradient analysis. Or one can elect the option to compute gradients
at a successful conclusion of the analysis at a given load step. In either case, STAGS will compute the second equation in (L.4): it will in turn examine the fabrications involved in the gradient definition, and zero out the $C$ matrix for all non-involved fabrications. For those involved in the variation, only a given $\{ij,ji\}$ pair of $C$ matrix components will be nonzero, and their values will be set to unity. An internal force calculation will be done (our system first variation), and a right hand side belonging to component $ij$ will be the result. After we have all 21 or 24 independent right hand sides, we compute their gradient basis vector $u_{ij}$ counterparts with the previously factored stiffness matrix $K$. If a current stiffness matrix is not available, one will be computed and factored. The basis vectors will be archived, along with the gradient of the displacements with respect to the load, something already required for path-following algorithms. At that point, S2 has finished its business.

The user now has the option of computing a particular displacement gradient by applying the first equation in (L.4). Let us for the moment assume that he has at hand the full set of coefficients of variation of $C$ with respect to his particular choice of design variables. We currently have an option in stapl to permit the user to input these coefficients as part of his interview or his.pin input stream. stapl will then sum up the contributions from each gradient basis vector and display the result. The user also has the option to display the gradient of strain resulting from this displacement gradient. In addition, one can display the displacement or strain gradient with respect to load. This latter information can be very revealing to an engineer wishing to determine what margins exist at a critical load, for instance. The plot will show where in the structure displacements or strains are most sensitive and perhaps critical in determining the outcome of a design. Lastly, we have the option of using previously computed eigenvectors as displacement gradients. In a very real sense, eigenvectors show alternative load paths that might be important should the stability of the structure go critical. The strain gradients from such an analysis will point out where strain is most likely to grow if stability is lost.

The GradC Program

For illustrative and reference purposes, a FORTRAN-language program (called GradC) that computes a $C$ matrix and its partial derivatives is listed in the following paragraphs—along with 6 subroutines that are called by it and with a FORTRAN header file (gccom.h) that is used by GradC and most of its subordinate routines. The logic and usage of this program are discussed after those listings. Output from GradC can be plotted by STAGS’ stapl processor and used as input for other applications.
Main Program (GradC):

************************************************************************
** Program GradC **
************************************************************************
#include "keydefs.h"

_implicit_none_
#
#include "gccom.h"
#
#include "pie.h"

Logical prompt

pi  = 3.141592653589793d0
dtr = pi/180.d0
rtd = 180.d0/pi
tol = 1.e-5

* GET MATERIAL DATA
* ================
10 print 100
20 print 200
read*, nmat

if (nmat.le.0) then
  print 300
  stop
endif

if (nmat.gt.MAXMAT) then
  print 400, MAXMAT
  goto 20
endif

do 30 i=1,nmat
  print 500, i
  read*, e1(i), nu12(i), g12(i), e2(i)
30 continue

* GET LAYER DATA
* ===============
40 print 600
read*, layers

if (layers.lt.1 .or. layers.gt.MAXLAYER) then
  print 700, MAXLAYER
  goto 40
endif

do 60 i=1,layers

50       print 800, i
       read*, imat(i), t(i), angle(i)
       if (imat(i).lt.1 .or. imat(i).gt.nmat) then
           print 900, imat(i)
           goto 50
       endif

60    continue

print 1000
read*, ecz

* COMPUTE AND PRINT C MATRIX
* ==========================
call compcc (e1,     nu12, g12, e2,    ecz,
&             layers, imat, t,   angle, cct)

print 1100
call out

* COMPUTE C MATRIX MATERIAL DERIVATIVE
* =====================================
70 if (prompt('Compute a C matrix material derivative? ')) then
    call MatDeriv
    goto 70
endif

* COMPUTE C MATRIX THICKNESS DERIVATIVE
* ======================================
80 if (prompt('Compute a C matrix thickness derivative? ')) then
    call ThickDeriv
    goto 80
endif

* COMPUTE C MATRIX ANGLE DERIVATIVE
* =================================
90 if (prompt('Compute a C matrix angle derivative? ')) then
    call AngleDeriv
    goto 90
endif
*     CHECK FOR ANOTHER C MATRIX COMPUTATION
*     ===============================================================
goto 10

100   format(/'----------------'
&       /'Welcome to GradC'
&       /'----------------')
200   format(/'Number of materials (zero to exit)? ',',$)
300   format(/'----------------'
&       /'GradC Completed'
&       /'----------------')
400   format('>>>  Number of materials limited to:',i4)
500   format('Material',i3,':  E1, Nu12, G12, E2? ',',$)
600   format('/'Number of layers? ','$,)
700   format('>>>  Number of layers limited to:',i4)
800   format('Layer',i4,':  Material, Thickness, Angle? ',',$)
900   format('>>>  Bad material number:',i4)
1000  format('/'Eccentricity? ','$,)
1100  format('/'C Matrix'/)

end

Subroutine AngleDeriv:

********************************************************************
Subroutine AngleDeriv
********************************************************************

_implicit_none_
#
include "gccom.h"

*     COMPUTE C MATRIX ANGLE DERIVATIVE
*     =================================
call zerof (MAXLAYER, dthick)
call zerof (MAXLAYER, dangle)

rtol = tol

do 10 i=1,layers

   print 100, i
   read*, dangle(i)

   if (dangle(i).ne.0) then
      rtol = max( rtol, abs(angle(i))*tol )
   endif

10   continue
10    continue

    print 200
    call LayerDeriv

100   format('Layer',i4,':  dAngle? ',$)
200   format(/'C Matrix Angle Derivative'/)
    end

Subroutine LayerDeriv:

************************************************************************
Subroutine LayerDeriv
************************************************************************

 Implicit_none_

#   include "gcocm.h"

*   COMPUTE A LAYER (THICKNESS OR ANGLE) DERIVATIVE
*   ===============================================

do 10 i=1,layers
   tp(i) = t(i) + dthick(i) * rtol
   anglep(i) = angle(i) + dangle(i) * rtol
10    continue

call compcc (e1, nu12, g12, e2, ecz, & layers, imat, tp, anglep, cctp)

do 20 i=1,layers
   tp(i) = t(i) - dthick(i) * rtol
   anglep(i) = angle(i) - dangle(i) * rtol
20    continue

call compcc (e1, nu12, g12, e2, ecz, & layers, imat, tp, anglep, cctm)

do 30 i=1,21
   cct(i) = (cctp(i)-cctm(i)) / (2.d0*rtol)
30    continue

call out
end
Subroutine MatDeriv:

************************************************************************
Subroutine MatDeriv
************************************************************************

implicit none

!     include "gccom.h"

Character eg*3

*     COMPUTE C MATRIX MATERIAL (E1, E2, OR G12) DERIVATIVE
*     ==============================================================

call zerof (MAXMAT, de1 )
call zerof (MAXMAT, de2 )
call zerof (MAXMAT, dg12)

10 print 100
read*, mat

if (mat.lt.1 .or. mat.gt.nmat) then
   print 200, mat
   goto 10
endif

20 print 300
read*, eg

call upper (3,eg,eg)

if (eg.eq.'E1') then
de1(mat) = 1.d0
print 400
elseif (eg.eq.'E2') then
de2(mat) = 1.d0
print 500
elseif (eg.eq.'G12') then
dg12(mat) = 1.d0
print 600
else
   print 700
   goto 20
endif

*     COMPUTE C MATRIX FINITE DIFFERENCE
*     ==============================================================
rtol = e1(mat) * tol

do 30 i=1,nmat
\[ e_{1p}(i) = e_1(i) + \text{de}_1(i) \times \text{rtol} \]
\[ e_{2p}(i) = e_2(i) + \text{de}_2(i) \times \text{rtol} \]
\[ g_{12p}(i) = g_{12}(i) + \text{dg}_{12}(i) \times \text{rtol} \]

30 continue

\[
\text{call compcc} \ (e_{1p}, \ \text{nu}_{12}, \ g_{12p}, \ e_{2p}, \ \text{ecz}, \ \& \ \text{layers}, \ \text{imat}, \ t, \ \text{angle}, \ \text{cctp})
\]

do 40 i=1,nmat
\[
\begin{align*}
  e_{1p}(i) &= e_1(i) - \text{de}_1(i) \times \text{rtol} \\
  e_{2p}(i) &= e_2(i) - \text{de}_2(i) \times \text{rtol} \\
  g_{12p}(i) &= g_{12}(i) - \text{dg}_{12}(i) \times \text{rtol}
\end{align*}
\]
40 continue

\[
\text{call compcc} \ (e_{1p}, \ \text{nu}_{12}, \ g_{12p}, \ e_{2p}, \ \text{ecz}, \ \& \ \text{layers}, \ \text{imat}, \ t, \ \text{angle}, \ \text{cctm})
\]

do 50 i=1,21
\[
\text{cct}(i) = (\text{cctp}(i) - \text{cctm}(i)) / (2.0 \times \text{rtol})
\]
50 continue

call out

100 format('Material number? ',$)
200 format('>>> Bad material number:',i4)
300 format('Derivative with respect to: E1, E2, or G12? ',$)
400 format('/C Matrix E1 Derivative'/)
500 format('/C Matrix E2 Derivative'/)
600 format('/C Matrix G12 Derivative'/)
700 format('>>> Enter: E1, E2, or G12')

end

Subroutine Out:

************************************************************************
Subroutine Out
************************************************************************

_implicit_none_
#
#include "gccom.h"

Integer ij
Integer j

*  -----------------------
ij(i,j) = i*(i-1)/2 + j
*  -----------------------
do 10 i=1,6
    print 100, (cct(ij(i,j)),j=1,i)
10    continue
100   format(1x,1p6e13.5)
end

Logical Function Prompt:

************************************************************************
Logical Function Prompt( msg)
************************************************************************

implicit none
Character msg(*)(*)
Character answer*3
Logical cmatch
print 10, msg
read*, answer
prompt = cmatch(answer,'Y^ES')
10   format(/a,$)
end

Subroutine ThickDeriv:

************************************************************************
Subroutine ThickDeriv
************************************************************************

implicit none
#
include "gccom.h"
call zerof (MAXLAYER, dthick)
call zerof (MAXLAYER, dangle)
rtol = tol
do 10 i=1,layers

   print 100, i
   read*, dthick(i)

   if (dthick(i).ne.0) then
      rtol = max( rtol, abs(t(i))*tol )
   endif

10    continue

print 200

call LayerDeriv

100   format('Layer',i4,: dThickness? ',',$)
200   format(/'C Matrix Thickness Derivative'/)

end

Fortran Header File gccom.h:

*
*     ====================
*     GLOBAL DATA: gccom.h
*     ====================

Integer    MAXMAT
Parameter (MAXMAT = 10)

Integer    MAXLAYER
Parameter (MAXLAYER = 100)

.Float_ angle (MAXLAYER)
.Float_ anglep(MAXLAYER)
.Float_ cct (21)
.Float_ cctm(21)
.Float_ cctp(21)
.Float_ dangle(MAXLAYER)
.Float_ de1 (MAXMAT)
.Float_ de2 (MAXMAT)
.Float_ dg12(MAXMAT)
.Float_ dthick(MAXLAYER)
.Float_ e1  (MAXMAT)
.Float_ e1p(MAXMAT)
.Float_ e2  (MAXMAT)
.Float_ e2p(MAXMAT)
.Float_ ecz
.Float_ g12 (MAXMAT)
Using the GradC Program

The GradC program allows users to compute a \( \mathbf{C} \) matrix and the partial derivatives of a \( \mathbf{C} \) matrix with respect to one of the material properties: \( E_1, E_2, \) and \( G_{12} \), or one of the laminate properties: thickness and orientation angles. The four logical stages provided by GradC are shown the following Figure:
Logical flow of program GradC.

Computation of a C Matrix

A C matrix is computed during the first stage of the GradC program. As shown in this Figure, during this stage the user supplies material and fabrication properties for a composite laminate. For each material in the laminate, the user supplies $E_1$, $Nu_{12}$, $E_2$, and $G_{12}$. For each layer, the user supplies material, thickness, and angle. Then, eccentricity is read and the C matrix is computed.
G12. This version of GradC permits up to 10 materials to be specified. This limit can be modified easily by changing the \texttt{MAXMAT} parameter in the \texttt{gccom.h} header file. Laminate geometry is next given by specifying the thickness and material principle axis orientation angle for each layer. This version of GradC permits up to 100 layers to be specified. This limit can be modified easily by changing the \texttt{MAXLAYER} parameter in the \texttt{gccom.h} header file. Next, the eccentricity of the shell middle surface from the element reference surface, as defined by the element nodes, is specified. GradC computes a C matrix using these data.

After a C matrix is computed, the user is asked if another C matrix computation is desired. If the response is yes, then the data just described are entered for another laminate. Otherwise, the user is moved along to the second stage of GradC.

\section*{Computation of a C Matrix Material Partial Derivative}

A C matrix material partial derivative is computed during the second stage of the GradC program. As shown in the Figure, during this stage the user identifies a material (as a number that identifies one of the materials entered during stage one) and the material property for the partial derivative. To identify the material property, the user enters one of the case insensitive values: ‘E1’, ‘E2’, or ‘G12’ (without quotes). Using these data, GradC computes the partial derivative of the C matrix computed in stage one with respect to the specified material property.

After a C matrix material partial derivative is computed, the user is asked if another material partial derivative computation is desired. If the response is yes, then the data just described are entered to specify another material and material property. Otherwise, the user is moved along to the third stage of GradC.

\section*{Computation of a C Matrix Thickness Partial Derivative}

A C matrix thickness partial derivative is computed during the third stage of the GradC program. As shown in the Figure, during this stage the user identifies each layer that is to be considered during the derivative computation. If a thickness difference coefficient (referred to by the code as \texttt{dThickness}) is specified (usually as 1), then the layer thickness is considered during the derivative computation. However, if a thickness difference coefficient is specified as 0, then the layer thickness is not considered during the derivative
computation. Using these data, GradC computes the partial derivative of the C matrix computed in stage one with respect to the specified layer thicknesses.

After a C matrix thickness partial derivative is computed, the user is asked if another thickness partial derivative computation is desired. If the response is yes, then the data just described are entered to specify another set of thickness difference coefficients. Otherwise, the user is moved along to the fourth stage of GradC.

**Computation of a C Matrix Angle Partial Derivative**

A C matrix angle partial derivative is computed during the fourth stage of the GradC program. As shown in the Figure, during this stage the user identifies each layer that is to be considered during the derivative computation. If an angle difference coefficient (referred to by the code as dTheta) is specified (usually as 1 or -1), then the layer angle is considered during the derivative computation. However, if an angle difference coefficient is specified as 0, then the layer angle is not considered during the derivative computation. Using these data, GradC computes the partial derivative of the C matrix computed in stage one with respect to the specified layer angles.

After a C matrix angle partial derivative is computed, the user is asked if another angle partial derivative computation is desired. If the response is yes, then the data just described are entered to specify another set of angle difference coefficients. Otherwise, the user is moved back to the first stage of GradC.
This appendix is not up-to-date vis a viz version 5.0 of the STAGS program. It is included in this document for reference purposes.

Introduction

The PATRAN to STAGS translator, PAT2S, illustrates the use of the STAR Data Put (DP) routines. PAT2S creates or updates a STAGS database. After preparation, the database is suitable input for STAGS’ S2 processor. Figure 4 shows the data flow from a PATRAN neutral file to a STAGS database. The following paragraphs describe the preparation of a STAGS database from a PATRAN neutral file.

Input File

The user first generates a PATRAN model. The user names the neutral file, case.pat.i. The user must also proved a user instruction file, case.usri. The user will replace case with a prefix appropriate for his model. i is an integer, which distinguishes files with the same prefix. PAT2S opens the case.pat.i or case.usri file with the greatest integer suffix. If gaps occur in the list of integer suffixes, PAT2S opens the last file before the gap occurs.

Execution

The user runs PAT2S in either of the following ways. The user can invoke STAGS’ STP processor with the case name as a command line argument and, then, choose the PATRAN to STAGS translator option. The user can also directly invoke PAT2S according to the following command line example,

```
% PAT2S case [debug]
```
case is the case name prefix. debug is an optional command line argument. If debug is present, the file case.sot.i contains a complete ASCII representation of data written into the STAGS database, case.sav. Look at the section, Output Files, for more information on these PAT2S output files. However, the ASCII information may be difficult to understand for users not familiar with the format of the STAGS database. The main purpose of the debug option is to provide a rough trace of the progress of translation in case of translation error.
Output Files

**PAT2S** uses the **STAGS** Access Routines (**STAR**) to place data into the **STAGS** database. Therefore, the output files for **PAT2S** are the same output files for **STAR**. These files include `case.sot.i` and `case.sav`. The prefix, `case`, and the suffix, `i`, follow the convention of the input files. `case.sot.i` contains diagnostic messages during translation. If the translation from **PATRAN** to **STAGS** is not successful, users should read the end of the `case.sot.i` file. **STAGS**’ S2 processor uses the `case.sav` file as input. `case.sav` is the model database. In most cases, **PAT2S** creates a new `case.sav` file. However, **PAT2S** can augment existing `case.sav` files. Because of this feature, the user is responsible for keeping copies of old `case.sav` files before translation.

User Instruction File Format

The user instruction file for **PAT2S** consists of a sequence of logical free-form input records, one logical record corresponding to each FORTRAN input list. In the following, a record is referred to by use of a name, such as “B-1 Element Type Translation”.

A free-form input record contains a number of **numerical data fields**, each separated by one of several **data terminators**, and optional **comments**, placed after the end of input data and ignored by the program. There are two types of numerical data fields, **integer** and **floating point**.

- Unless otherwise noted, an entry on a data record is an integer if the variable name starts with I–N. Otherwise it is to be treated as a floating point number.
- Integer field widths cannot exceed 5 characters, including the optional sign. Leading blanks and a terminating character are not counted.
- Floating point numbers must contain a decimal point; an exponent is optional. The exponent may be in any form allowed by regular FORTRAN “E” type format. Thus, the following data forms are all equivalent

  1450. 1.45E+03 .145+4 1.45E3
• Floating-point decimal field widths are unlimited, however floating-point data are truncated to about 7 or 8 decimal digits.

• A numerical data field may begin with any number of blanks (which are ignored) and is terminated either with the last column for input data (column 80) or with one of the following characters:

  " " blank  
  "," comma  
  "/" slash  
  "$" dollar sign

• Blanks are ignored everywhere on records containing numerical data, except when they occur between two numerical data fields; then the first field is terminated by the blank. Thus input integers or floating point numbers may never contain embedded blanks.

**WARNING:** A blank line is not ignored; it produces a single logical record.

• A comma is normally used to terminate a data field when the logical record contains additional data. Hence, when the last data field on one line is terminated with a comma, this functions as a continuation character and indicates that the logical record continues with the next input line. Otherwise, the logical record is terminated when the end of data on the line is reached. Successive commas on a line may be used to generate blank values of integer items in a list. For example, "20,,10" is equivalent to "20, 0, 0, 10", which is equivalent to "20 0 0 10".

• A slash terminates a logical record, and means that any following data field begins a new logical record. Thus, several logical records may be constructed with one input line (whereas with the use of commas, several input records might comprise a single logical record). A slash following a comma prevents continuation and the comma then functions as a simple terminator.

• A dollar sign in column 1 indicates a comment card.

A dollar sign in any other column signals the end of data on a record and means that the remaining information on the line is ignored. This space may be used for comments. A dollar sign may be used also when the data field on a line is terminated by a comma, *i.e.*, when the logical record continues on the following input line.
User Instruction File Records

Because STAGS has modeling capabilities which the PATRAN neutral file cannot describe, a separate set of user instructions must accompany the PATRAN neutral file. The following pages describe the input records in a User Instruction file, which is subject to the above free form guidelines.
A-1 Case Title

The case title is read on the first line, which may contain any alphanumeric character. Subsequently any number of comment records can be added provided they begin with a “$” in column 1. Comments can also be included at the end of a data line, a “$” terminating data, and the comment following. A list of the complete input file, including any comment records, is printed at the beginning of all text output files. The user is urged to use this record as a way to document the analysis.

COMMENT

COMMENT case title

  go to B-1
**B-1 Element Type Translation**

The next set of lines map the shape, nodes, and configuration number in the PATRAN neutral file (Packet Type 02, Variables *IV*, *NODES*, and *CONFIG*) to a corresponding element type in STAGS. There is a default set of mappings as indicated below.

---

**NECODE**

Number of user defined element translation records. 0 means use the default. A 0 indicates that the translation will proceed according to the following map:

<table>
<thead>
<tr>
<th>Shape</th>
<th>Nodes</th>
<th>Configuration Number</th>
<th>STAGS element type</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>210</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>211</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>110</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>130</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>250</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>320</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0</td>
<td>410</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
<td>411</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>0</td>
<td>480</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>1</td>
<td>510</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>2</td>
<td>710</td>
</tr>
</tbody>
</table>

The elements 130 and 250 are not currently supported.

```plaintext
if (NECODE > 0) go to B-1a
else    go to C-1
```
B-1a Element Type List

This record is read **NECODE** times. The following applies only if **NECODE** is not 0. Each line has 4 integers: **ISHAPE**, **INODES**, **IPAT**, and **ISTAGS**.

**ISHAPE(i) INODES(i) IPAT(i) ISTAGS(i)**

**ISHAPE(i)**  PATRAN neutral file (Packet Type 02), variable **IV**.

**INODES(i)**  PATRAN neutral file (Packet Type 02), variable **NODES**.

**IPAT(i)**  PATRAN neutral file (Packet Type 02), variable **CONFIG**.

**ISTAGS(i)**  STAGS element type (e.g. 320, 410, or 480).

*go to C-1*
C-1 Material Translation

The next record associates a PATRAN material ID (Packet Type 02, variable PID) with a STAGS material number. This record is currently not active. Material information in the PATRAN neutral file (Packet type 03) is not translated into STAGS material properties information. The user should place a 0 for this record.

NMATCD

NMATCD | Number of user defined associations between PATRAN material properties and STAGS material properties. 0 means the PATRAN material ID corresponds to the STAGS material ID. Since this record is not active, the user should always enter 0.

if (NMATCD > 0) go to C-1a
else  go to D-1
**C-1a Material List**

This record is read `NMATCD` times. The following applies only if `NMATCD` is not 0. Each line has 2 integers: **MPAT** and **MSTAGS**.

<table>
<thead>
<tr>
<th>MPAT(i)</th>
<th>MSTAGS(i)</th>
</tr>
</thead>
</table>

**MPAT(i)**
- **PATRAN** neutral file (Packet Type 02), variable **PID**.

**MSTAGS(i)**
- **STAGS** material ID number.

`go to D-1`
D-1 Element Property

PATRAN can associate an arbitrary list (Packet Type 04, Variable DATA) of numbers with an element. PATRAN tags each element with a property ID (Packet Type 02, Variable PID) in order to establish this association. PAT2S uses the element property ID to specify mount table ID, Cross Section Table ID, or Wall Fabrication Table ID. The actual mount table, Cross Section Table, or Wall Fabrication Table occur in subsequent lines of the user instruction file. PAT2S also uses the PATRAN element property list of numbers to specify the parameters normally specified in the STAGS T-1a through T-4c records.

NELMCD

| NELMCD | Number of user defined associations between PATRAN element property ID and STAGS mount table ID, Cross Section Table ID, or Wall Fabrication Table ID. If this record contains 0, then the default association is PATRAN element property ID equals STAGS table ID. |

if (NELMCD > 0) go to D-1a
else go to E-1
D-1a Element Property List

This record is read NELMCD times. The following applies only if NELMCD is not 0. Each line has 2 integers: IEPAT and IESTAGS.

For mount elements, the PATRAN element property list (Packet Type 04, variable DATA) contains the following:

MATID, IMNT1, IMNT2, LINKS, RLX1, RLY1, RLZ1, RLX2, RLY2, RLZ2

MATID is the PATRAN material property ID, which PAT2S does not use. PATRAN assigns this number. The other variables occur in the STAGS T-1a and T-1b input records. The user should consult the STAGS user manual for the meaning of these entries. For hyperelastic fastener elements, the PATRAN element property list (Packet Type 04, variable DATA) contains the following:

MATID, IMNT1, IMNT2, IMNT3, IMNT4, IMNT5, IMNT6

These variables occur in the STAGS T-1C input record. For beam elements, the PATRAN element property list (Packet Type 04, variable DATA) contains:

MATID, XSI, ECY, ECZ, ILIN, IPLAS

These variables occur in the STAGS T-2 input record. For shell elements, the PATRAN element property list contains the following:

MATID, UNUSED, ZETA, ECZ, ILIN, IPLAS, INTEG, IPENL, IANG, RX, RY, RZ

The variable, UNUSED, must contain 0. The other variables correspond to fields in the STAGS T-3a through T-4c records for triangles or quadrilaterals. Users should consult the STAGS User Manual for the meaning of these records.

The wall reference vector, (RX, RY, RZ), may differ from element to element. Each distinct wall reference vector requires a distinct Packet Type 04 in the PATRAN neutral file. Because specifying a wall reference vector in this manner is laborious, PAT2S offers another method to indicate the wall reference vector. The user triggers this alternate method when IANG=1 and (RX, RY, RZ)=(0.0, 0.0, 0.0) in the element property list for triangles and quadrilaterals.

For this alternate method to work the user must do the following. The user first specifies alternate coordinate systems (Packet Type 05) in PATRAN and, then, associates a particular coordinate system with a particular element of the model. PATRAN puts the alternate system information in
the coordinate frame data (Packet Type 05). PATRAN projects the x-axis of a Cartesian frame or the r-axis of a cylindrical or spherical frame onto the plane of the element. This projected axis is the PATRAN material axis. The $\theta_1$ angle in the element data (Packet Type 02) is the angle between the PATRAN material axis and the vector from node 1 to node 2 of the element. Since we do not know the algorithm used by PATRAN to determine this material axis, we assume that this material axis lies in the same plane as the x-y plane of the STAGS element frame. PAT2S transforms this material axis to the PATRAN global frame. The resulting vector is the wall reference vector ($\mathbf{RX}, \mathbf{RY}, \mathbf{RZ}$). Users should consult the PATRAN manual on Mesh Generation for more information.

IEPAT(i) IESTAGS(i)

<table>
<thead>
<tr>
<th>IEPAT(i)</th>
<th>PATRAN neutral file (Packet Type 02), variable PID.</th>
</tr>
</thead>
<tbody>
<tr>
<td>IESTAGS(i)</td>
<td>STAGS mount table ID, Cross Section Table ID, or Wall Fabrication Table ID.</td>
</tr>
</tbody>
</table>
**E-1 Load Translation**

**PATRAN** allows two types of element loading: line loads and surface tractions. **PATRAN** also allows node forces. In addition to these loads and tractions, **STAGS** also permits live and dead pressure. Live pressure remains normal to the deformed surface throughout geometrically nonlinear deformations. Dead pressure is constant (i.e. normal to the undeformed configuration). Whereas **PATRAN** defines element loading in the parametric directions of the parent geometry, **STAGS** has five options for defining element distributed load directions:

- element-edge coordinates \((x_e, y_e, z_e)\)
- element-surface coordinates \((x_s, y_s, z_s)\)
- shell coordinates \((X', Y', Z')\)
- global coordinates \((x_g, y_g, z_g)\)
- element normal \(z'\)

**STAGS** also defines two point load directions:

- computational coordinates \((x'', y'', z'')\)
- global coordinates \((x_g, y_g, z_g)\)

In order to resolve these conflicts and to allow full use of **STAGS** element loading features, we established rules for interpreting **PATRAN** distributed loads (Packet Type 06), and **PATRAN** node forces (Packet Type 07).

In order to facilitate distributed loading and point force interpretation, we established six load categories with specific load type and coordinate direction combinations. The load categories with the associated load types and coordinate directions are the following:

- **Intrinsic:**
  - line loads in element-edge coordinates
  - surface tractions in element-surface coordinates
  - node forces in computational coordinates
- **Shell:**
  - line loads and surface tractions in shell coordinates
- **Global:**
  - line loads and surface tractions in global coordinates
  - node forces in global coordinates
- **Live:**
  - live pressure in the \(z'\) (element normal) direction
- **Dead:**
  - dead pressure in the \(z'\) (element normal) direction
velocity:
velocity dependent point load.

For the categories live and dead, we treat the $z'$, the 3rd load component as the pressure value. Other load components are irrelevant for those categories.

**PATRAN** groups distributed loads and node forces into load sets. Each identified by a unique load set ID (LSID). We use the LSID to categorize distributed loads and point forces according to the above six types. By default, all load sets are in the intrinsic category, which permits line loads in element-edge coordinates, surface tractions in element-surface coordinates, and node forces in computational coordinates. The user instruction file provides a means for independently placing each load set into one of the other five categories. Each LSID may appear in only one category.

With this background information, we now return to the next records in the user instruction file. The next lines in the user instruction file are the load specification records.

---

**NSHELL NGLOB NLIVE NDEAD NVELC**

**NSHELL**  
**NSHELL** must always equal 0. This option is currently not supported. If **NGLOB**, **NLIVE**, **NDEAD**, and **NVELC** are all 0, then **PAT2S** uses the intrinsic designation: line loads in element-edge frame, surface tractions in element-surface frame, node forces in computational frame.

**NGLOB**  
**NGLOB** is the number of IDs considered in the global frame. This number includes both distributed load set IDs and node forces IDs.

**NLIVE**  
**NLIVE** is the number of IDs considered as live loads. This number includes only distributed load set IDs.

**NDEAD**  
**NDEAD** is the number of IDs considered as dead loads. This number includes only distributed load set IDs.

**NVELC**  
**NVELC** is the number of IDs considered as velocity dependent loads. This number includes only point force set IDs.

if (NGLOB > 0) go to E-1a  
else if (NLIVE > 0) go to E-1b  
else if (NDEAD > 0) go to E-1c  
else if (NVELC > 0) go to E-1d  
else go to F-1
E-1a Global Frame List

This record is read \texttt{NGLOB} times.

\textbf{NLGLOB(i)}

\texttt{NLGLOB(i) PATRAN} neutral file (Packet Type 06 or 07), variable \texttt{IV}.

if \((\text{NLIVE} > 0)\) go to E-1b
else if \((\text{NDEAD} > 0)\) go to E-1c
else if \((\text{NVELC} > 0)\) go to E-1d
else go to F-1
E-1b Live Load List

This record is read \textit{NLIVE} times.

\begin{tabular}{l}
\textbf{NLLIVE(i)} \\
\hline
\end{tabular}

\begin{itemize}
  \item \textbf{NLLIVE(i)} \quad \textit{PATRAN} neutral file (Packet Type 06), variable \textit{IV}.
  \item \textbf{if} \ (\textbf{NDEAD} \ > \ 0) \ \textit{go to E-1c}
  \item else if \ (\textbf{NVELC} \ > \ 0) \ \textit{go to E-1d}
  \item else \ \textit{go to F-1}
\end{itemize}
**E-1c Dead Load List**

This record is read **NDEAD** times.

---

<table>
<thead>
<tr>
<th><strong>NLDEAD(i)</strong></th>
<th>PATRAN neutral file (Packet Type 06), variable <strong>IV</strong>.</th>
</tr>
</thead>
</table>

- if (NVELC > 0) *go to* E-1d
- else *go to* F-1
E-1d Velocity Dependent List

This record is read NVELC times.

<table>
<thead>
<tr>
<th>NVELC(i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PATRAN neutral file (Packet Type 07), variable IV.</td>
</tr>
</tbody>
</table>

$\Rightarrow$ go to F-1
Appendix X

**F-1 Load System**

**PATRAN** groups load data using an arbitrary number of load sets, which **PATRAN** identifies by a unique load set ID (LSID). **STAGS** organizes loads into two load systems, A and B. In this context, “load” includes both distributed (element) loads and point (nodal) forces. By default, we place all loads in load system A. The next user instruction lines allow users to override this default by permitting them to place each LSID in load system B or to omit each LSID from the **STAGS** model.

---

**NSYSB NOMIT**

**NSYSB** is the number of IDs indicating use of load system B. Load system A is the default.

**NOMIT** is the number of IDs omitted from translation.

\[
\text{if (} \text{NSYSB} > 0) \text{ go to F-1a} \\
\text{else if (} \text{NOMIT} > 0) \text{ go to F-1b} \\
\text{else go to G-1}
\]
F-1a System B List

This record is read $\text{NSYSB}$ times.

$\text{NLSYSB(i)}$

$\text{NLSYSB(i)}$ PATRAN neutral file (Packet Type 06 or 07), variable $\text{IV}$.

if ($\text{NOMIT} > 0$) go to F-1b
else go to G-1
F-1b Omitted Load List

This record is read \texttt{NOMIT} times.

\begin{verbatim}
\texttt{NLOMIT(i)}
\end{verbatim}

\begin{verbatim}
\texttt{NLOMIT(i)} \quad \texttt{PATRAN} neutral file (Packet Type 06 or 07), variable \texttt{IV}.
\end{verbatim}

\begin{verbatim}
\textbullet\quad \textit{go to G-1}
\end{verbatim}
**G-1 Constraint Translation**

Prescribed nodal displacements are translated from **PATRAN** node displacements records (Packets Type 08). **PATRAN** groups this data using an arbitrary number of constraint sets, each identified by a unique constraint set ID (CSID). **STAGS** assembles prescribed nodal displacements into its two load systems, A and B.

Closely related to prescribed nodal displacements are boundary conditions, which are a set of degree-of-freedom (DOF) flags indicating the condition of each DOF as free (unconstrained) or fixed (constrained).

**STAGS** also recognizes initial conditions, initial displacement, or initial velocity. We make use of the **PATRAN** node displacements record to also specify initial displacement or initial velocity. With these numerous possible interpretations of the **PATRAN** node displacements record, we introduce the constraint translation records in the next lines of the user instruction file.

---

**NDBASC NDINCR NINTDS NINTVL**

- **NDBASC** is the number of IDs indicating basic boundary conditions.
- **NDINCR** is the number of IDs indicating incremental boundary conditions.
- **NINTDS** is the number of IDs indicating initial displacement.
- **NINTVL** is the number of IDs indicating initial velocity.

```plaintext
if (NDBASC > 0) go to G-1a
else if (NDINCR > 0) go to G-1b
else if (NINTDS > 0) go to G-1c
else if (NINTVL > 0) go to G-1d
else go to H-1
```
G-1a Basic Boundary Condition List

This record is read NDBASC times.

<table>
<thead>
<tr>
<th>IDBASC(i)</th>
<th>PATRAN neutral file (Packet Type 08), variable IV.</th>
</tr>
</thead>
</table>

     if (NDINCR > 0) go to G-1b
     else if (NINTDS > 0) go to G-1c
     else if (NINTVL > 0) go to G-1d
     else go to H-1
This record is read `NDINCR` times.

<table>
<thead>
<tr>
<th>IDINCR(i)</th>
<th>PATRAN neutral file (Packet Type 08), variable IV.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>if <code>(NINTDS &gt; 0)</code> go to G-1c</td>
</tr>
<tr>
<td></td>
<td>else if <code>(NINTVL &gt; 0)</code> go to G-1d</td>
</tr>
<tr>
<td></td>
<td>else go to H-1</td>
</tr>
</tbody>
</table>
**G-1c Initial Displacement List**

This record is read \textbf{NINTDS} times.

\begin{verbatim}
IINTDS(i) PATRAN neutral file (Packet Type 08), variable IV.

if (NINTVL > 0) go to G-1d
else go to H-1
\end{verbatim}
G-1d Initial Velocity List

This record is read \texttt{NINTVL} times.

\begin{verbatim}
IINTVL(i)
\end{verbatim}

\begin{verbatim}
IINTVL(i) \quad \texttt{PATRAN} neutral file (Packet Type 08), variable \texttt{IV}.
\end{verbatim}

\begin{verbatim}
\textit{go to H-1}
\end{verbatim}
H-1  Constraint System Translation

PATRAN groups nodal displacements using an arbitrary number of constraint set IDs (CSID). STAGS organizes constraints into two load systems, A and B. By default, we place all constraints in load system A. The next user instruction lines allow users to override this default by permitting them to place each CSID in load system B or to omit each CSID from the STAGS model.

NDSYSB NDOMIT

NDSYSB  NDSYSB is the number of IDs indicating use of load system B. Load system A is the default.

NDOMIT  NDOMIT is the number of IDs omitted from translation.

if (NDSYSB > 0) go to H-1a
else if (NDOMIT > 0) go to H-1b
else go to I-1
**H-1a System B List**

This record is read `NDSYSB` times.

<table>
<thead>
<tr>
<th>IDSYSB(i)</th>
<th>PATRAN neutral file (Packet Type 08), variable IV.</th>
</tr>
</thead>
</table>

- if `(NDOMIT > 0)` *go to* H-1b
- else *go to* I-1
H-1b Omitted Constraint List

This record is read NDOMIT times.

IDOMIT(i)

IDOMIT(i) PATRAN neutral file (Packet Type 08), variable IV.

* go to l-1
**I-1 Multipoint Constraint Translation**

Multiple interpretation of the PATRAN multipoint constraint record (Packet Type 14) is also possible. We use the PATRAN multipoint constraint record to specify either a STAGS multipoint constraint, Lagrangian constraint, or partial compatibility constraint. The next set of lines from the user instruction file removes the ambiguity of interpreting multipoint constraint records.

---

<table>
<thead>
<tr>
<th>NCMPC</th>
<th>NCLGC</th>
<th>NCPCM</th>
<th>NCOMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCMPC</td>
<td>NCLGC</td>
<td>NCPCM</td>
<td>NCOMIT</td>
</tr>
</tbody>
</table>

**NCMPC**
- **NCMPC** is the number of multipoint constraint IDs for interpretation as multipoint constraints.

**NCLGC**
- **NCLGC** is the number of multipoint constraint IDs for interpretation as Lagrangian constraint.

**NCPCM**
- **NCPCM** is the number of multipoint constraint IDs for interpretation as partial compatibility constraints.

**NCOMIT**
- **NCOMIT** is the number of multipoint constraint IDs omitted from translation.

```plaintext
if (NCMPC > 0) go to l-1a
else if (NCLGC > 0) go to l-1b
else if (NCPCM > 0) go to l-1c
else if (NCOMIT > 0) go to l-1d
else go to J-1
```
I-1a Multipoint Constraint List

This record is read $\text{NCMPC}$ times.

\[
\text{MCMPC}(i)
\]

\[
\begin{array}{l}
\text{MCMPC}(i) \quad \text{PATRAN neutral file (Packet Type 14), variable IV.} \\
\text{if (NCLGC > 0) go to I-1b} \\
\text{else if (NCPM > 0) go to I-1c} \\
\text{else if (NCOMIT > 0) go to I-1d} \\
\text{else go to J-1}
\end{array}
\]
I-1b Lagrangian Constraint List

This record is read NCLGC times.

MCLGC(i)

\begin{itemize}
\item \textbf{MCLGC(i)} \quad \text{PATRAN neutral file (Packet Type 14), variable IV.}
\item if (NCPCM > 0) \textcolor{red}{go to I-1c}
\item else if (NCOMIT > 0) \textcolor{red}{go to I-1d}
\item else \textcolor{red}{go to J-1}
\end{itemize}
I-1c Partial Compatibility Constraint List

This record is read $\text{NCPCM}$ times.

$\text{MCPCM}(i)$

$\text{MCPCM}(i)$ \hspace{1em} \text{PATRAN neutral file (Packet Type 14), variable IV.}$

- if ($\text{NCOMIT} > 0$) go to I-1d
- else go to J-1
I-1d Omitted Multipoint Constraint List

This record is read NCOMIT times.

MCOMIT(i)

MCOMIT(i)      PATRAN neutral file (Packet Type 14), variable IV.

جو      go to J-1
J-1 Master-Slave Translation

Since PATRAN cannot specify master-slave node relationships, we use the user instruction file to establish these relationships between nodes. The next set of lines in the user instruction file is for master slave relationships.

NMASLAV

NMASLAV is the number of master-slave relationships between nodes.

if (NMASLAV > 0) go to J-1a
else go to K-1
J-1a Master-Slave List

This record is read **NMASLAV** times.

<table>
<thead>
<tr>
<th>NSUNT(i) NSNODE(i) NMUNT(i) NMNODE(i)</th>
</tr>
</thead>
</table>

**NSUNT(i)**  
NSUNT(i) is the unit number of the slave node. If NSUNT(i) = 0, then NSNODE(i) is a global node number.

**NSNODE(i)**  
NSNODE(i) is slave node number in the unit. If NSUNT(i) = 0, then slave node number is a global node number.

**NMUNT**  
NMUNT(i) is the unit number of the master node. If NMUNT(i) = 0, then NMNODE(i) is a global node number.

**NMNODE**  
NMNODE(i) is master node number in the unit. If NMUNT(i) = 0, then master node number is a global node number.

*go to K-1*
**K-1 Tables Translation**

Because we currently do not translate the material property data records (packet type 03) in *PATRAN* to a *STAGS* database, we use the user instruction file to specify the material properties of the model. We make use of the format in the B-3, l-1, l-2, l-3, l-3a, l-4a, l-4b, l-4c, l-4d, J-1, J-2a, J-2b, J-3a, J-3b, K-1, K-2, K-5a, K-5b, and K-5c *STAGS* input records to specify the material properties, plasticity, creep, mount element force tables, beam element Cross Section Tables, and shell Wall Fabrication Tables. Users should consult the *STAGS* User Manual for more information on these input records.

<table>
<thead>
<tr>
<th>NTAM</th>
<th>NTAB</th>
<th>NTAW</th>
<th>NTAP</th>
<th>NTAMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTAM</td>
<td>Refer to record B-3, Data Table Summary, in the <em>STAGS</em> User Manual for the meaning of these variables.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NTAB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NTAW</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NTAP</td>
<td>Although this variable is used in record B-3, it has no meaning for <em>PAT2S</em>. Users should set it to 0.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NTAMT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```plaintext
if (NTAM > 0) go to L-1
else if (NTAB > 0) go to M-1
else if (NTAW > 0) go to N-1
else if (NTAMT > 0) go to O-1
else go to P-1
```
L-1–L-4 Material Properties

These records are read $\text{NTAM}$ times. The subsequent input records are identical with STAGS’ l-1, l-2, l-3, and l-3a input records which specify the material elastic, plastic, and creep properties. Users should consult the STAGS User Manual for more information on these input records.

```
if (NTAB > 0) go to M-1
else if (NTAW > 0) go to N-1
else if (NTAMT > 0) go to O-1
else go to P-1
```
M-1–M-4 Table Information

These records are read $NTAB$ times. The subsequent input records are identical with STAGS’ I-4a, I-4b, I-4c, and I-4d input records which specify the mount element table information. Users should consult the STAGS User Manual for more information on these input records.

```plaintext
if (NTAW > 0) go to N-1
else if (NTAMT > 0) go to O-1
else go to P-1
```
N-1–N-5 Beam Cross Section Properties

These records are read \texttt{NTAW} times. The subsequent input records are identical with STAGS’ J-1, J-2a, J-2b, J-3a, and J-3b input records which specify the beam cross section information. Records J-4a and J-4b are not supported by \texttt{PAT2S}. Users should consult the STAGS User Manual for more information on these input records.

\begin{verbatim}
if (NTAMT > 0) go to O-1
else go to P-1
\end{verbatim}
O-1–O-5 Shell Wall Properties

These records are read \texttt{NTAMT} times. The subsequent input records are identical with \texttt{STAGS’ K-1}, K-2, K-5a, K-5b, and K-5c input records which specify the beam cross section information. Records K-3a, K-3b, K-4a, K-4b, and K-6 are not supported by \texttt{PAT2S}. Users should consult the \texttt{STAGS} User Manual for more information on these input records.

\textit{go to P-1}
P-1 Output Control

The final line in the user instruction file is identical to the STAGS V-1 input record, the element unit output control record. The variables NSELD and NSELS are not active. The user should set these variables to 0. Users should consult the STAGS User Manual for more information on these input records.

End of user instructions

Translation Details

The following gives additional details on the translation of a PATRAN neutral file to a STAGS database. Figure 5 and Figure 6 associate PATRAN data records to STAR put routines. The user instruction file which directs PAT2S in processing a PATRAN neutral file determines some of the parameters to these STAR put routines. The tail of each arrow in the figures is a PATRAN data packet. The head of each arrow point to the associated STAR put routine, which we used to create an entry in the STAGS database.

Node Data Translation

Figure 5 indicates that the translation of node data (Packet Type 01) also utilizes coordinate frame information (Packet Type 05). A node references an alternate coordinate frame using the PATRAN variable, CID (Packet Type 01). The PATRAN alternate coordinate frame is a rectangular, cylindrical, or spherical frame. This frame serves the same function as the STAGS branch coordinate frame for shells. This frame determines a node’s computational frame to global frame transformation. If IV = 0 (Packet Type 05), then PATRAN does not specify an alternate frame. The node’s computational to global frame transformation is the identity transformation. If IV = 1, then PATRAN specifies a rectangular alternate frame. The node’s computational to global frame transformation is the variable, R (Packet Type 05). If IV = 2, then PATRAN specifies a cylindrical frame. The computational frame at the node has the PATRAN convention with the x-axis as the radial axis, the y-axis as the circumferential axis, and the z-axis as the axial axis. We construct a transformation from the PATRAN alternate frame to this computational frame. The node’s computational to global transformation combines the transformation from computational to alternate frame with the alternate to global frame transformation in the variable, R. If IV = 3, then PATRAN specifies
a spherical frame. The computational frame at the node has the PATRAN convention with the x-axis as the radial axis, the y-axis as the azimuthal axis, and the z-axis as the longitudinal
Appendix X

We construct a transformation from the PATRAN alternate frame to this computational frame. The computational to global frame transformation for the node combines the computational to alternate frame transformation with the alternate to global frame transformation in the variable, \( \mathbf{R} \).

**Element Data Translation**

Figure 5 indicates that the translation of element data (Packet Type 02) also uses the element properties information (packet type 04). The description of input record B-1 in the user instruction file explains the default interpretation of the `CONFIG` variable (Packet Type 02). This interpretation maps PATRAN element types to STAGS element types. In the D-1A input record, we described the necessary information for the `DATA` field of the element properties packet (Packet Type 04), which gives attributes of individual elements.

There are a few special elements which the user can describe by a PATRAN neutral file. If the element shape equals 2, \( \mathbf{V} = 2 \) (Packet Type 02), and the element configuration equals
7, CONFIG = 7, then the element is an added mass element. The first node in LNODES array is the location of the added mass and the second entry in DATA (Packet Type 04) is the added mass value. If IV = 4, NODES = 9, and CONFIG = 1, then the element is the STAGS 510 transition element. If IV = 4, NODES = 9, and CONFIG = 2, then the element is the STAGS 710 transition element. For these two elements, some entries in LNODES have values less than or equal to 0 in order to reflect node pattern in the transition element. Since these elements are not part of the standard PATRAN element libraries, interactive use of PATRAN cannot generate these elements. However, other programs generating PATRAN neutral files can easily generate a 510 or 710 element entry.

Distributed Load Translation

Depending on the entries in the user instruction file, the translator interprets the distributed loads (Packet Type 06) in the PATRAN neutral file. If the user instructs PAT2S to interpret a distributed load as a live pressure, then PAT2S uses only every third component in PDATA. PAT2S calls the routine dppress. If the user instructs PAT2S to interpret a distributed load as a dead pressure, then PAT2S uses only every third component in PDATA. PAT2S puts a distributed line load in the STAGS database with a call to dpline and a distributed surface load in the STAGS database with a call to dptrae. Although these translation mechanisms are in place, STAGS S2 processor currently does not recognize distributed traction loads. Any attempt to interpret distributed surface loads will invoke dptrae. dptrae will then exit with an error.

Node Displacement Translation

Depending on the entries in the user instruction file, the translator interprets the node displacement (Packet Type 08) in the PATRAN neutral file. If the user instructs PAT2S to interpret a node displacement as a basic boundary condition or incremental boundary condition, then PAT2S ignores the load values in DDATA and uses only the displacement component flags in ICOMP. The PATRAN convention for the displacement component flags is 0 for free degrees of freedom and 1 for constrained degrees of freedom. The STAGS convention is the opposite. Therefore, PAT2S changes the contents of the ICOMP variable from the PATRAN convention to the STAGS convention. The contents of the ICOMP variable adds to the single point constraint specified by the PSPC variable in the node data packet (Packet Type 01). PAT2S then calls the routine dpbcci to create an entry in the STAGS database.

If the user instructs PAT2S to interpret a node displacement as an initial displacement or an initial velocity, PAT2S uses the value in DDATA and calls the routine dpitcon to create an
entry in the STAGS database. PAT2S calls dpadis for the default action of applying a specified displacement at the node.

**Multipoint Constraint Translation**

Depending on the entries in the user instruction file, the translator interprets the multipoint constraint data (Packet Type 14) in the PATRAN neutral file. If the user instructs PAT2S to interpret a multipoint constraint data set as a Lagrangian constraint, then PAT2S calls the routine dplag. If the user instructs PAT2S to interpret a multipoint constraint data set as a partial compatibility constraint, then PAT2S calls the routine dpprcm. The PATRAN dependent node becomes the STAGS slave node and the PATRAN independent node becomes the STAGS master node. PAT2S ignores all but the first independent node. If the user does not provide a translation interpretation, PAT2S’ default action is to translate PATRAN’s multipoint constraint packet into a STAGS multipoint constraint and by calling the routine, dpmpc.