DRAFT

ESPRIT Project 322

CAD*I

CAD Interfaces

SPECIFICATION FOR EXCHANGE OF PRODUCT ANALYSIS DATA

VERSION 3

Edited By :
Deborah Thomas
Jan Van Maanen
Michael Mead

For Working Group 6:

Gesellschaft für Strukturanalyse, Aachen, West Germany
NEH Consulting Engineers ApS, Broendby, Denmark
Rutherford Appleton Laboratory, Chilton, UK
ERDISA, Madrid, Spain
INITEC, Madrid, Spain

31 October 1988

Acknowledgement

We would like to acknowledge the assistance of the Commission of the European Communities in helping to fund the work described in this report.

Abstract

One of the aims of ESPRIT project 322 on CAD Interfaces (CAD*I) is the development of standards for data exchange between Product Definition (Computer Aided Design) systems and Product Analysis (Finite Element and Computer Aided Testing) systems.

This report describes a standard for a neutral file which can be used for exchange of data between Finite Element preprocessors, Finite Element analysis systems and Finite Element postprocessors. The lexical structure and syntax of the file are formally defined. The semantics are defined in terms of a reference model which is a formal description of all the entities that can occur in the neutral file with their attributes. Additionally, an informal description of all structures appearing in the neutral file is given.

The report also looks at the problems of transferring geometry data from Computer Aided Design systems to Finite Element preprocessors and proposes some heuristic rules for the transformations which are needed in this transfer.

List of Contributors

Carlos Arias, INITEC

Emilio Baños, INITEC

Bernhard Carl, GfS

Bryan F Colyer, RAL

Luis Delgado, ERDISA

Rafael Fernàndez, ERDISA

Niels E Hansen, NEH

Horst Heinrichs, NEH

Helmut Helpenstein, GfS

Jes M Jessen, NEH

Troels Ladefoged, NEH

Michael Mead, RAL

Wilfried Melder, GfS

Alan Middleton, RAL

Javier Moldes, INITEC

Miguel A Moreno, INITEC

Jose L Navarro, ERDISA

Manuel Peralta, ERDISA

Axel M Spliid, NEH

Deborah Thomas, RAL

Jan Van Maanen, RAL

Contents

A General Information
A.1 Executive Summary....................1
A.1.1 General Introduction....................1
A.1.2 Objectives....................1
A.1.3 Achievements....................2
A.1.4 Contribution to CAD*I....................2
A.1.5 Description of the Report....................3
A.2 Background Information....................5
A.2.1 Exchange of Product Analysis Data....................5
A.2.2 Existing Standards....................5
A.2.3 Approach to Design of Specification....................6
A.3 Example Neutral File....................9
A.4 General File Structure....................15
A.4.1 Product Definition and Product Analysis Domains....................15
A.4.2 Interaction between File and Processors....................16
A.4.3 Internal Organisation of the File....................17
A.5 External Envelope Concept....................19

B Reference Section
B.1 Syntax Definition....................21
B.1.1 The Alphabet....................21
B.1.2 Token Definitions....................22
B.1.3 BNF Description of the File....................24
B.1.4 Description of EXPRESS for FE Specification....................24
B.1.4.1 Entities and Attributes....................25
B.1.4.2 Base Types....................25
B.1.4.3 Entity References....................25
B.1.4.4 Enumeration....................25
B.1.4.5 Unique and Optional....................26
B.1.4.6 Aggregates....................26
B.1.4.7 Other definition constructs....................26
B.1.4.8 Constraints....................27
B.2 Levels of the Neutral File....................29
B.3 Keywords for FE Model definition....................31
B.3.1 OPENMODEL....................31
B.3.2 UNITS....................32
B.3.3 COORD....................33
B.3.4 FREEDOM....................35
B.3.5 NODE....................36
B.3.6 ISO....................37
B.3.7 ISOFULL....................38
B.3.8 ANISO2D....................39
B.3.9 ANISO3D....................41
B.3.10 MATERIAL....................43
B.3.11 PBAR....................44
B.3.12 BEAMG....................45
B.3.13 BEAMI....................46
B.3.14 BEAML....................47
B.3.15 BEAMU....................48
B.3.16 BEAMT....................49
B.3.17 BEAMO....................50
B.3.18 BEAMR....................51
B.3.19 STIFF....................52
B.3.20 DAMP....................53
B.3.21 PMASS....................54

B.3.22 THICK....55
B.3.23 THICKV....56
B.3.24 PROPERTY....57
B.3.25 OR1....58
B.3.26 OR3....58
B.3.27 ORIENT....59
B.3.28 MASS....60
B.3.29 DAMPER....62
B.3.30 SPRING....63
B.3.31 BAR....64
B.3.32 BEAM2....65
B.3.33 TRI3....67
B.3.34 TRI6....68
B.3.35 TRI7V....69
B.3.36 QUAD4....71
B.3.37 QUAD8....72
B.3.38 QUAD9V....73
B.3.39 TETRA4....75
B.3.40 TETRA10....76
B.3.41 TETRA15V....77
B.3.42 PENTA6....79
B.3.43 PENTA15....80
B.3.44 PENTA21V....81
B.3.45 HEXA8....83
B.3.46 HEXA20....84
B.3.47 HEXA27V....85
B.3.48 CLOSEMODEL....86
B.4 Keywords for FE Analysis definition....87
B.4.1 OPENANA....87
B.4.2 COLLECT....88
B.4.3 CDOFLIST....89
B.4.4 SCOND....90
B.4.5 DCOND....91
B.4.6 COUPLELIST....92
B.4.7 NOCOUPLE....93
B.4.8 COPY....94
B.4.9 ASSEM....95
B.4.10 TRN....96
B.4.11 TRE....97
B.4.12 ALLNODES....98
B.4.13 ALLELEMS....98
B.4.14 NODELIST....99
B.4.15 ELEMLIST....100
B.4.16 NODAL....101
B.4.17 ELEMENTAL....102
B.4.18 PRESET1....103
B.4.19 PRESET2....104
B.4.20 MPC....105
B.4.21 TEMP....106
B.4.22 LOADP1....107
B.4.23 LOADP2....107
B.4.24 FREQRG....108
B.4.25 STEADY....108
B.4.26 IMPULS....109
B.4.27 RECTAN....109
B.4.28 SAWTOO....110
B.4.29 SINUS....111

B.4.30 SWEEPL....................112
B.4.31 SWEEPE....................113
B.4.32 STARTL....................114
B.4.33 STARTE....................115
B.4.34 RANDOM....................116
B.4.35 DYNLOAD....................117
B.4.36 DISCRLOAD....................118
B.4.37 NODELOAD....................119
B.4.38 LOADL1....................120
B.4.39 LOADL2....................120
B.4.40 EDGELOAD....................121
B.4.41 LOADS1....................122
B.4.42 LOADS2....................122
B.4.43 SURFLOAD....................123
B.4.44 LOADV1....................124
B.4.45 LOADV2....................124
B.4.46 ELEMLOAD....................125
B.4.47 GRAVLOAD....................126
B.4.48 TEMPLOAD....................127
B.4.49 CONVEC....................128
B.4.50 FLUX....................129
B.4.51 QVOL....................130
B.4.52 QVOLP....................131
B.4.53 FREQRANGE....................132
B.4.54 TIMERANGE....................133
B.4.55 MDCONST....................134
B.4.56 MODALDAMP....................135
B.4.57 ANSTATIC....................136
B.4.58 ANREIGV....................137
B.4.59 ANCEIGV....................138
B.4.60 ANFRESP....................139
B.4.61 ANTRESP....................140
B.4.62 ANSTEMP....................141
B.4.63 OUTDISP....................142
B.4.64 OUTSTRESS....................143
B.4.65 OUTSTRAIN....................144
B.4.66 OUTFORCE....................145
B.4.67 FIRSTFR....................146
B.4.68 MODELIST....................147
B.4.69 FREQOUT....................147
B.4.70 OUTEIGFR....................148
B.4.71 OUTMODESH....................149
B.4.72 OUTTEMP....................150
B.4.73 OUTFLUX....................151
B.4.74 CLOSEANA....................152
B.5 Keywords for FE Results definition....................153
B.5.1 OPENFERES....................153
B.5.2 ANCASE....................154
B.5.3 DISPR....................155
B.5.4 DISPC....................156
B.5.5 STRENOR....................157
B.5.6 STRENOC....................158
B.5.7 STREEQR....................159
B.5.8 STREEQC....................159
B.5.9 STREPRR....................160
B.5.10 STREPRC....................161
B.5.11 STREENR....................162

B.5.12 STREENC....163
B.5.13 STREEIR....164
B.5.14 STREEIC....165
B.5.15 STRANOR....166
B.5.16 STRANOC....167
B.5.17 STRAEQR....168
B.5.18 STRAEQC....169
B.5.19 STRAPRR....170
B.5.20 STRAPRC....171
B.5.21 STRAENR....172
B.5.22 STRAENC....173
B.5.23 STRAEIR....174
B.5.24 STRAEIC....175
B.5.25 NODFORCER....176
B.5.26 NODFORCEC....177
B.5.27 EIGFR....178
B.5.28 DAMPRAT....178
B.5.29 NODTEMP....179
B.5.30 NODFLUX....180
B.5.31 RC....181
B.5.32 MATSEQ....182
B.5.33 MATBAND....183
B.5.34 MATGEN....184
B.5.35 MATSHP....185
B.5.36 MATDEF....186
B.5.37 MATVAL....187
B.5.38 CLOSEFERES....188
B.6 Notes....189
B.7 Keyword Cross-reference....191

C Implementation
C.1 Processors for FE data....195
C.2 Software developed by the Partners....197
C.2.1 Software developed by NEH....197
C.2.1.1 Program File Handler....197
C.2.1.2 Program NEH61....198
C.2.1.3 Program NEH62....198
C.2.1.4 Program NEH63....199
C.2.2 Software developed by RAL....201
C.2.2.1 Program RDNF....201
C.2.2.2 PRENAS....201
C.2.2.3 PRE-NF....202
C.2.2.4 NF2FG....202
C.2.3 Software developed by GfS....204
C.2.3.1 Program NFUTIL....204
C.2.3.2 Program FEREAD....204
C.2.3.3 Program FEINW....205
C.2.3.4 Program FEOUTW....206
C.2.4 Software developed by ERDISA....208
C.2.4.1 Program CADMOD....208
C.2.5 Software developed by INITEC....209
C.2.5.1 Program IENF....209
C.2.5.2 Program CVNF....209
C.2.5.3 Program NFAN....210

D Discussion of the CAD-FEM Data Transfer
D.1 Intoduction....211

D.2 CAD/FEM Interchange Environment....................................213
D.3 The CAD*I approach to the CAD/FEM interface....................217
D.4 Guidelines for FE modelling...219
D.4.1 The complexities involved...219
D.4.1.1 The requirements of the solution 219
D.4.1.2 The nature of the problem 220
D.4.1.3 The mechanics of the problem 220
D.4.1.4 The geometry 221
D.4.1.5 The loads 221
D.4.1.6 The computer code 222
D.4.2 Advanced FE Modelling..222
D.4.2.1 The state of the art of automatic FE modelling ..222
D.4.2.2 An expert system for the CAD/FEM interface 225
D.5 Heuristic rules for FE Modelling....................................227
D.5.1 Development of a model of heuristic rules set..........227
D.5.1.1 Heuristic Rule Set 1: Model regions 227
D.5.1.2 Heuristic Rule Set 2: Required overall accuracy .229
D.5.1.3 Heuristic Rule Set 3: Type of analysis 230
D.5.1.4 Heuristic Rule Set 4: Type of problem 231
D.5.1.5 Heuristic Rule Set 5: Geometry type 232
D.5.1.6 Heuristic Rule Set 6: Geometry complexity 234
D.5.1.7 Heuristic Rule Set 7: Load time distribution 235
D.5.1.8 Heuristic Rule Set 8: Program analysis capabilities 235
D.5.2 Suggestions on how the heuristic rules might be implemented in an expert system 235
D.6 The FEM to CAD interface...237
D.7 Conclusions...239

References..241

List of Illustrations

1. Model of a Cantilever Beam using Volume Elements........... 10
2. Deformed Configuration of the Cantilever Beam.............. 10
3. CAD and Analysis Domains.................................... 15
4. Interaction Between Neutral File and Applications.......... 16
5. Structure of Neutral File.................................. 17
6. Local Coordinate System for 1D elements.................... 61
7. Node Ordering for Triangular 2D elements................... 66
8. Node Ordering for Quadrilateral 2D elements................ 70
9. Node Ordering for Tetrahedron solid elements............... 74
10. Node Ordering for Pentahedron solid elements............... 78
11. Node Ordering for Hexahedron solid elements................ 82
12. Interfaces to FE processors................................ 196
13. Reducing the number of Interfaces.......................... 196
14. Software developed by the partners......................... 200
15. Connection between Modelling and Analysis.................. 215

A General Information

A.1 Executive Summary

A.1.1 General Introduction

One major goal of ESPRIT Project No 322, CAD-Interfaces (CAD*I) is the development of techniques for the exchange of product analysis data.

This report describes a set of specifications for the exchange of product analysis data. A syntax for representing the analysis data in a neutral file is presented; this syntax is formally described in Backus-Naur form. A reference model for the finite element (FE) analysis data is given which uses a formal data modelling language to define the fundamental data entities, their attributes and interrelations. Syntactical expressions (statements) for entities in these reference models are worked out. The overall internal structure of the neutral file, and the interaction between the neutral file and the computing environment via pre- and post-processors are presented.

The final part of this report presents an investigation into the problems of transferring geometry data between CAD and FE systems and proposes some initial heuristic rules.

A.1.2 Objectives

This report presents the results obtained by Working Group 6 of the CAD*I project team. The objectives of Working Group 6 are:

1. development of a specification for a neutral file for exchange of product analysis data, suitable for describing analytical data.

2. development of pre- and post-processors for a number of representative FE pre-processors, FE analysis codes and FE post-processors.

3. benchmarking the developed techniques by exchanging suitable test files.

4. contributing to the international standardization activities.

5. development of heuristic rules for the transformation of CAD geometry into FE geometry.

A.1.3 Achievements.

A file for exchange of product analysis data stores a combination of information that is used or generated at different stages of the analysis process. Considering FE analysis, one can distinguish FE pre-processing data, FE model and analysis data, and FE post-processing data. These different processes may be implemented in different programs, possibly in different computing environments.

The achievements that are reported on in this report are:

1. A definition of the syntax for the files. This syntax has been discussed during various meetings with the Working Groups in the CAD*I project that are developing techniques for exchange of product definition data (in terms of wire frame, surfaces and constructive solid geometry models).

2. A reference model for the data entities that are relevant in analytical product analysis and syntactical expressions for the data entities.

3. A lay-out of concepts on the overall structure of the files and interaction of the files with the computing environment via pre- and post-processors.

4. Investigation into heuristic rules for the transformation of CAD geometry to FE geometry.

The partners in Working Group 6 have implemented the file definitions and written pre- and post-processors which act as interfaces between the neutral file and application programs. The post-processors make use of LR(1), a parser generator and a parser written in standard Fortran, and also a lexical analyser. The use of these software tools has been very successful and has convinced the partners in Working Group 6 that they represent valuable aids for implementing the required processors.

A.1.4 Contribution to CAD*I

The prime objective of the CAD*I project is to develop interfaces for transfer of product data between the different processes of the CAD environment. This environment is often considered to include processes, such as FE analysis and CAT and is therefore sometimes referred to by the term Computer Aided Engineering (CAE). It is therefore essential that interfaces allow for transfer of both product definition data and product analysis data.

The work that is reported on in this paper concentrates on the aspect of exchange of product analysis data. With the aim of having a highly unified set of specifications for both product definition and analysis data, the work has not been performed independently from the Working Groups active in the area of specifications for exchange of product definition data but rather in close collaboration. The results of this essential interaction are reflected in various parts of this report, so that the results that are reported on here truly contribute to the prime objective of the CAD*I project.

A.1.5 Description of the Report

This report describes the work in three major sections. Section A contains an informal description of the results, background information (Chapter A.2), scope of the work (Chapter A.3), example neutral files (Chapter A.4) and an informal description of the file format (Chapters A.5 and A.6).

Section B is the reference section and constitutes the formal file specification. Section C deals with implementation issues, describes the pre- and post-processors that have been developed, and so on.

Section D contains a description of the work done on transformation of CAD geometry into geometry suitable for FE preprocessing.

4 Exchange of Product Analysis Data

A.2 Background Information

A.2.1 Exchange of Product Analysis Data

Product analysis data includes all the data which is necessary to perform an analysis on a product together with the results of that analysis. To illustrate this point consider the data required and generated by a finite element (FE) analysis.

First the geometry of the product may be generated using a CAD system and this can then be input to a pre-processor where the product is 'meshed' (split into elements). Within the pre-processor, material properties, physical properties, loads and constraints are added to the model. All this data is then passed to a suitable FE analysis program and the results are given to a post-processor for display.

The above sequence poses problems at several stages. Each of the four programs CAD, pre-processor, analysis and post-processor could exist on different machines. The data to be transferred at each stage will most probably be in different formats. The output from each program is usually not suitable as input for the next program and any interfaces that exist are specific to two particular programs only.

The objective of the CAD*I project is to define a technique by which this data can be stored and transferred so that it can be written or read by any of the programs. Only two interfaces need exist for each individual program - one to read the standard data and one to write it. In this way data can be transferred easily between different programs and different machines.

A.2.2 Existing Standards

The first task when the Project began was to review available existing standards. Three standards were considered by working group 6; ANSI Y14.26M-1981, IGES Version 1 and IGES Version 2. The IGES standard was originally designed for the transfer of engineering drawings.

ANSI Y14.26M-1981; a number of criticisms of this standard have been made; they were reviewed briefly in the Technical Annex to the Contract. The first part of the Standard, taken from IGES Version 1 is straightforward, though capable of defining only simple constructs. Difficulties arise from the second part; its relationship to the IGES file is by no means clear. The data formats are inefficient and voluminous. There is no specific way to deal with entities such as loads and constraints, other than as properties associated with geometric entities.

IGES: Version 2 attempts to meet the criticisms which were levelled at Version 1. It extends into new areas, of which finite element modelling is of interest here. However, the Standard would still be inadequate for our purposes. Version 3, which was published recently, has not been studied in detail by us.

The IGES Standard has also been reviewed in detail by working group 1, which reached similar conclusions.

During the development of the CAD*I specifications the IGES organisation decided to develop a new standard for product definition and analysis with the name PDES (Product Definition Exchange Standard). The International Standards Organisation subsequently decided not to develop a standard of their own but to collaborate with IGES in the development of the new standard. The new ISO standard, although functionally the same as PDES, is called STEP (Standard for the Exchange of Product data). Through the International Standards Organisation CAD*I has been able to exert a significant influence on the development of STEP.

A.2.3 Approach to Design of Specification.

As the intention is to provide a neutral file as part of a data base which may be accessed from various programs, it was decided to design a form which would be common to all files. To achieve this necessitates a method which is

- as general as possible

- extendible to cover future needs

The files must be transferable between computers of different makes and different operating systems: this requirement is most generally satisfied by files of ASCII characters. To ensure extendibility, the most satisfactory approach is to have files in a form analogous to modern structured programming languages: each entity in a file is defined by a keyword, followed by all pertinent information within delimiters (parentheses). To the program reading such a file, the keyword has the nature of an operator, which instructs the program how it should handle the information.

Software for writing such files gives no particular difficulty, but the writing of software to read a file would be a very time consuming and error-prone task. Fortunately, similar problems encountered by the designers and authors of programming language compilers (Fortran, Pascal etc.) have led to the development of generally available utilities for this task. The first of these is a program to generate lexical analysis routines, the second one a 'compiler compiler' which creates routines for parsing the source statements. Examples of such utilities are: LEX (lexical analyser generator) and YACC (parser generator) - both being components of UNIX, and LR(1) (a parser generator and parser written in standard Fortran).

It was clear from the outset that the only satisfactory way was to adopt formal methods for describing syntax. Only by these means is it possible to guarantee complete lack of ambiguity. The method used is the Backus-Naur Form (BNF). Most file structures, entities, etc. are described in two different ways in this document: one is an informal description which is understandable and reasonably accurate for the general user of the standards and another description is given using formal languages to prevent any ambiguities.

In the design of the specification great emphasis was put on a correct and formal description of the file syntax and the entities occurring in the file. The alphabet characters are carefully defined and from these tokens are defined using regular expressions. A context-free grammar is used to describe the syntactical structure of the file. The entities occurring in the file and their attributes and relationships are defined using a data modelling language called Express which is described in more detail in the reference section.

8 Exchange of Product Analysis Data

A.3 Example Neutral File

As an example outlining the structure of the neutral file the following model is provided. The component used for the demonstration of the structure of the neutral file is a cantilever beam.

The material is an isotropic linear elastic material. The two material constants used to define the constitutive relation are Young's modulus E and Poisson's ratio nue. Here, they are $E = 3.0 \times 10^{10}$ N/m^2 and nue = 0.25. Furthermore, the constant mass density rho is given as rho = 2600 kg/m^3. This information is given in the MATERIAL keyword.

The beam is subdivided into 48 hexahedron volume elements each with 8 nodes giving a total of 117 nodes. The beam is shown in Fig. 1. The beam is fixed at the right end of the beam, i.e. at nodes 109 to 117. The applied load is a node load at the left end of the beam, i.e. at nodes 6, 7 and 15, in the negative z-direction. The deformed configuration of the beam is shown in Fig. 2.

The above information is given in the OPENMODEL block. In the OPENANA block is the description of the required analysis. Here, we perform a static analysis and the requested output is the displacements for all nodes, the six stress components for all nodes, von Mises equivalent stress for all nodes and the principal stresses for all nodes.

The requested output is given in the OPENFERES block.

The corresponding neutral file is listed afterwards. However, some marked sections of the file are omitted for the purpose of presentation.

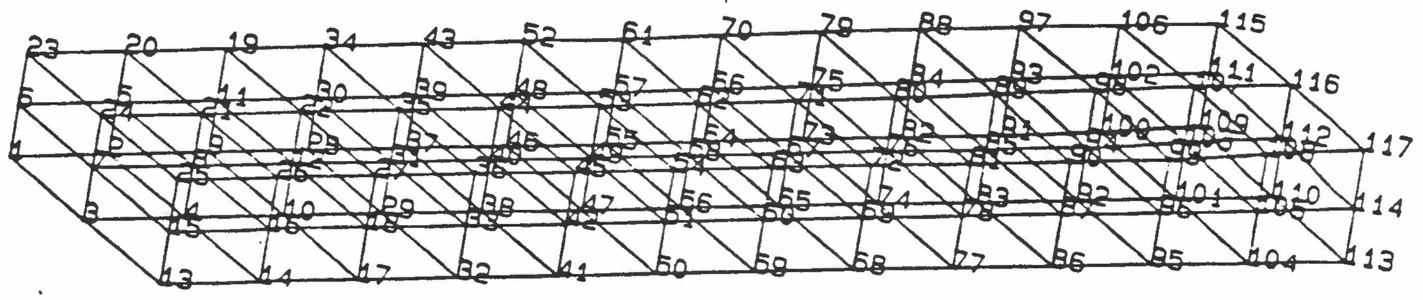


Fig. 1. Model of a Cantilever Beam Using Volume Elements.

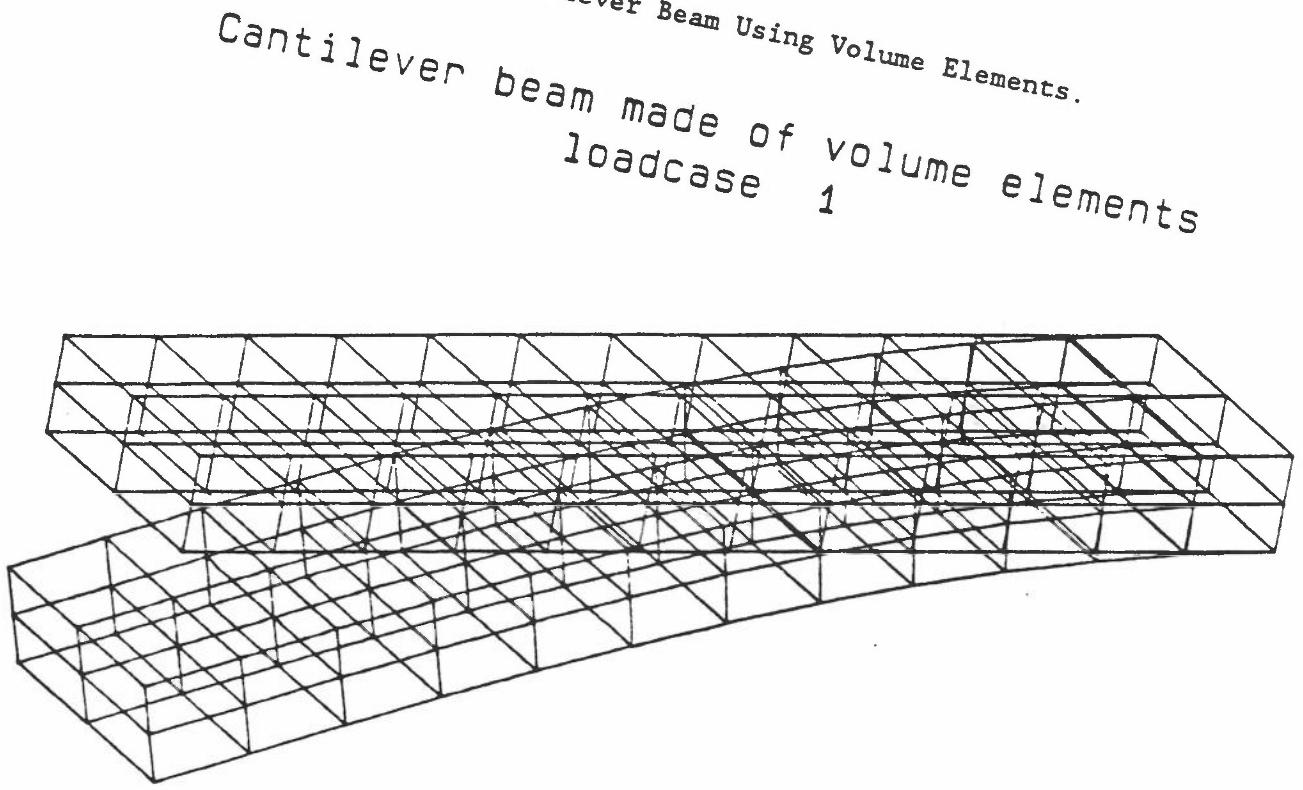


Fig. 2. Deformed Configuration of the Cantilever Beam

```
CAD*I_FORMAT_BEGIN_19851011  1987/06/03 16:18
CAD*I_FORMAT_BEGIN_19881031  1987/06/03 16:18
OPENMODEL (#Cantilever: ,'Cantilever beam made of volume elements');
UNITS (#Cantilever, 1., 1., 1., 1., 273.15, 1., 1.,360.);
FREEDOM (#F57:'456',);
FREEDOM (#F64:'123456',);
NODE (#N1:1, 1.2, 0., 0.,,#F57);
NODE (#N2:2, 1.1, 0., 0.,,#F57);
NODE (#N3:3, 1.2, 0.15, 0.,,#F57);
NODE (#N4:4, 1.1, 0.15, 0.,,#F57);
NODE (#N5:5, 1.1, 0., 0.05,,#F57);

omitted lines

NODE (#N113:113, 0., 0.3, 0.,,#F64);
NODE (#N114:114, 0., 0.3, 0.05,,#F64);
NODE (#N115:115, 0., 0., 0.1,,#F64);
NODE (#N116:116, 0., 0.15, 0.1,,#F64);
NODE (#N117:117, 0., 0.3, 0.1,,#F64);
MATERIAL (#M0:'Isotropic material',ISO (3.E+10, 0.25, 2600.));
HEXA8 (#E1:1,#M0,,#N109,#N100,#N101,#N110,#N111,#N102,
#N103,#N112);
HEXA8 (#E2:2,#M0,,#N110,#N101,#N104,#N113,#N112,#N103,
#N105,#N114);
HEXA8 (#E3:3,#M0,,#N101,#N92,#N95,#N104,#N103,#N94,#N96,
#N105);
HEXA8 (#E4:4,#M0,,#N100,#N91,#N92,#N101,#N102,#N93,#N94,
#N103);
HEXA8 (#E5:5,#M0,,#N111,#N102,#N103,#N112,#N115,#N106,
#N107,#N116);

omitted lines

HEXA8 (#E44:44,#M0,,#N5,#N6,#N7,#N8,#N20,#N23,#N24,#N21);
HEXA8 (#E45:45,#M0,,#N9,#N2,#N4,#N10,#N11,#N5,#N8,#N12);
HEXA8 (#E46:46,#M0,,#N10,#N4,#N14,#N17,#N12,#N8,#N16,
#N18);
HEXA8 (#E47:47,#M0,,#N4,#N3,#N13,#N14,#N8,#N7,#N15,#N16);
HEXA8 (#E48:48,#M0,,#N2,#N1,#N3,#N4,#N5,#N6,#N7,#N8);
CLOSEMODEL (#Cantilever);

OPENANA (#Anal:#Cantilever,'Static analysis');
COLLECT (#OB1:'Collect object',(#E1,#E2,#E3,#E4,#E5,#E6,
#E7,#E8,#E9,#E10,#E11,#E12,#E13,#E14,#E15,#E16,#E17,#E18,#E19,
#E20,#E21,#E22,#E23,#E24,#E25,#E26,#E27,#E28,#E29,
#E30,#E31,#E32,#E33,#E34,#E35,#E36,#E37,#E38,#E39,#E40,
#E41,#E42,#E43,#E44,#E45,#E46,#E47,#E48));
LOADP1 (#LP1:3,,-0.25);
LOADP1 (#LP2:3,,-0.5);
NODELOAD (#LP1,1,#N15);
NODELOAD (#LP2,1,#N7);
NODELOAD (#LP1,1,#N6);
ANSTATIC (#AN1:#OB1,(1));
ALLNODES (#ALL1:(#OB1));
OUTDISP (#AN1,#ALL1,);
OUTSTRESS (#AN1,NODAL(1,#ALL1,));
OUTSTRESS (#AN1,NODAL(2,#ALL1,));
OUTSTRESS (#AN1,NODAL(3,#ALL1,));
CLOSEANA (#Anal);

OPENFERES (#FERES1:#Anal,'1st. load case');
ANCASE (#AN1,STAT,1,);
DISPR (#N1,-4.741E-08,-2.403E-10,-7.556E-07, 0.,0., 0.,);
DISPR (#N2,-4.707E-08,-3.168E-10,-6.608E-07, 0.,0., 0.,);
DISPR (#N3,-4.738E-08,-2.579E-19,-7.560E-07, 0.,0., 0.,);
DISPR (#N4,-4.703E-08,-2.498E-19,-6.614E-07, 0.,0., 0.,);
DISPR (#N5,-9.355E-21,-3.579E-20,-6.608E-07, 0.,0., 0.,);

omitted lines

DISPR (#N113, 0., 0., 0., 0., 0., 0.,);
DISPR (#N114, 0., 0., 0., 0., 0., 0.,);
DISPR (#N115, 0., 0., 0., 0., 0., 0.,);
DISPR (#N116, 0., 0., 0., 0., 0., 0.,);
DISPR (#N117, 0., 0., 0., 0., 0., 0.,);
STRENOR (#N1,-50.6, 13.4,-7.08, 3.52,-33.3,-1.97,);
STRENOR (#N2,-100.3, 9.175,-2.57, 5.795,-33.3,-1.62,);
STRENOR (#N3,-50.6, 13.4,-7.08,-6.661E-16,-33.3,
3.331E-16,);
STRENOR (#N4,-100.3, 9.175,-2.57,-4.441E-16,-33.3,
7.401E-17,);
STRENOR (#N5, 2.368E-15,-5.921E-16, 0., 1.48E-16,-33.3,
-1.62,);

omitted lines

STRENOR (#N113,-1160.,-231.,-149., 64.7,-33.3,-18.7,);
STRENOR (#N114, 0., 0., 0., 0.,-33.3,-18.7,);
STRENOR (#N115, 1160., 231., 149., 64.7,-33.3, 18.7,);
STRENOR (#N116, 1160., 231., 149.,-1.421E-14,-33.3,
-3.553E-15,);
STRENOR (#N117, 1160., 231., 149.,-64.7,-33.3,-18.7,);
STREEQR (#N1, 81.1184);
STREEQR (#N2, 119.466);
STREEQR (#N3, 80.817);
STREEQR (#N4, 119.011);
STREEQR (#N5, 57.7455);

omitted lines

STREEQR (#N113, 981.263);
STREEQR (#N114, 66.1494);
STREEQR (#N115, 981.263);
STREEQR (#N116, 974.305);
STREEQR (#N117, 981.263);
STREPRR (#N1, 15.8298,-68.6761, 8.56632, 0.29046, 0.82028,
-0.49273, 0.87972 -0.02633, 0.47475, 0.37645,
-0.57136,-0.72927,);
STREPRR (#N2, 11.9039,-110.781, 5.18245, 0.21146, 0.79038,
-0.57497, 0.9551,-0.04218, 0.29328, 0.20755,
-0.61117,-0.7638,);
STREPRR (#N3, 13.4,-68.6192, 10.9392, 0.46157, 0.,
-0.8871, 0.87949, 0., 0.47591, 0.47591, 0.,-0.87949,);
STREPRR (#N4, 9.17493,-110.568, 7.69779, 0.29101, 0.,
-0.95672, 0.9556, 0., 0.29465, 0.29465, 0.,-0.9556,);
STREPRR (#N5, 33.3394,-33.3394, 4.487E-06, 0.70627, 0.03436,
-0.70711, 0.70627, 0.03436, 0.70711, 0.99882, 0.04859, 0.,);

omitted lines

STREPRR (#N113,-142.758,-1165.49,-231.754, 0.047, 0.23995,
-0.96965, 0.99716,-0.06841, 0.03141,-0.0588,
-0.96837,-0.24249,);
STREPRR (#N114, 38.1914,-38.1914, 5.14E-06, 0.61654, 0.34623,
-0.70711, 0.61654, 0.34623, 0.70711, 0.87192, 0.48964, 0.,);
STREPRR (#N115, 1165.49, 142.758, 231.754, 0.99716, 0.06841,
-0.03141, 0.047,-0.23995, 0.96965, 0.0588,
-0.96837,-0.24249,);
STREPRR (#N116, 1161.1, 147.905, 231., 0.99946, 0.,
-0.03288, 0.03288, 0., 0.99946, 0.03582, 0., 0.99936,);
STREPRR (#N117, 1165.49, 142.758, 231.754, 0.99716,
-0.06841,-0.03141, 0.047, 0.23995, 0.96965,-0.0588,
-0.96837, 0.24249,);
CLOSEFERES (#FERES1);
CAD*I_FORMAT_END___19881031
CAD*I_FORMAT_END___19851011
```

A.4 General File Structure

A.4.1 Product Definition and Product Analysis Domains

The CAD*I project is dealing with two data domains; CAD systems containing product definition data, and product analysis data in FEM and experimental systems. The two domains have an overlapping area which contains a geometrical description of an analytic model, see fig. 3. This is a subset of the complete CAD representation of the structure. In the analysis domain this subset produced by CAD is the basis for further processing.

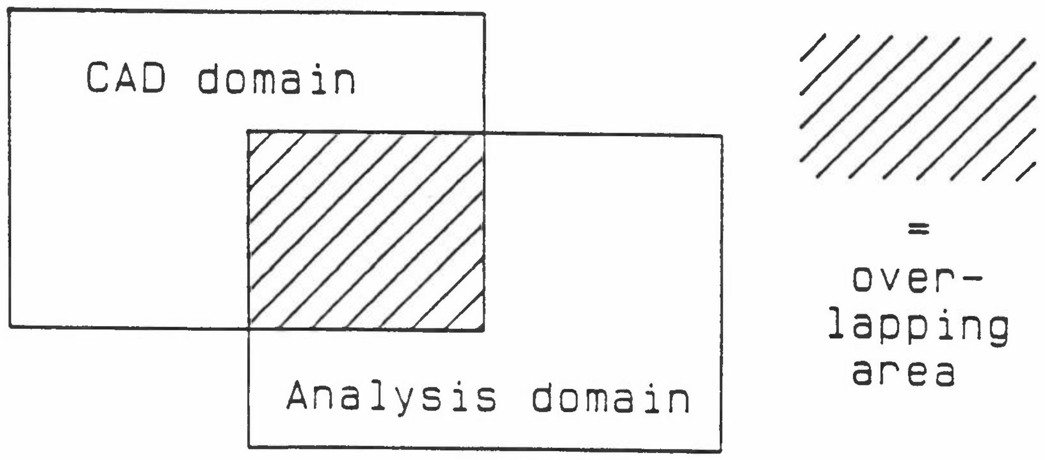


Fig. 3. CAD and Analysis Domains.

However, the FEM pre-processors often will require some structural changes to the model provided by the CAD system. The geometric description of a model, though part of the neutral file, is being designed by other working groups of the CAD*I project and is therefore not covered in this document. For the purpose of an analysis it is often required to change the geometry (e.g. removal of irrelevant detail). If required the new geometry can be described in the same neutral file. The description of the neutral file allows for the analysis to point to a certain geometry description.

This is important because the CAD side must be able to read that part of the file. So in the overlapping area all keywords and semantics have to be checked mutually, whereas in the other areas the definitions of the entities can be independent.

A.4.2 Interaction between File and Processors

The neutral file can contain all information beginning with the geometrical input in the overlapping area and ending with the optimization output. This makes it possible to have exactly one file for one model and in that way to enable easy comparison of results (e.g. between calculation and measurement).

There will be no data base management system for the manipulation of the neutral file. However, the interface to the applications program may be divided into an application-independent part reading and writing the neutral file and an application-dependent part which writes input files or reads output files or communicates with the data base of the applications program. This is shown in fig. 4.

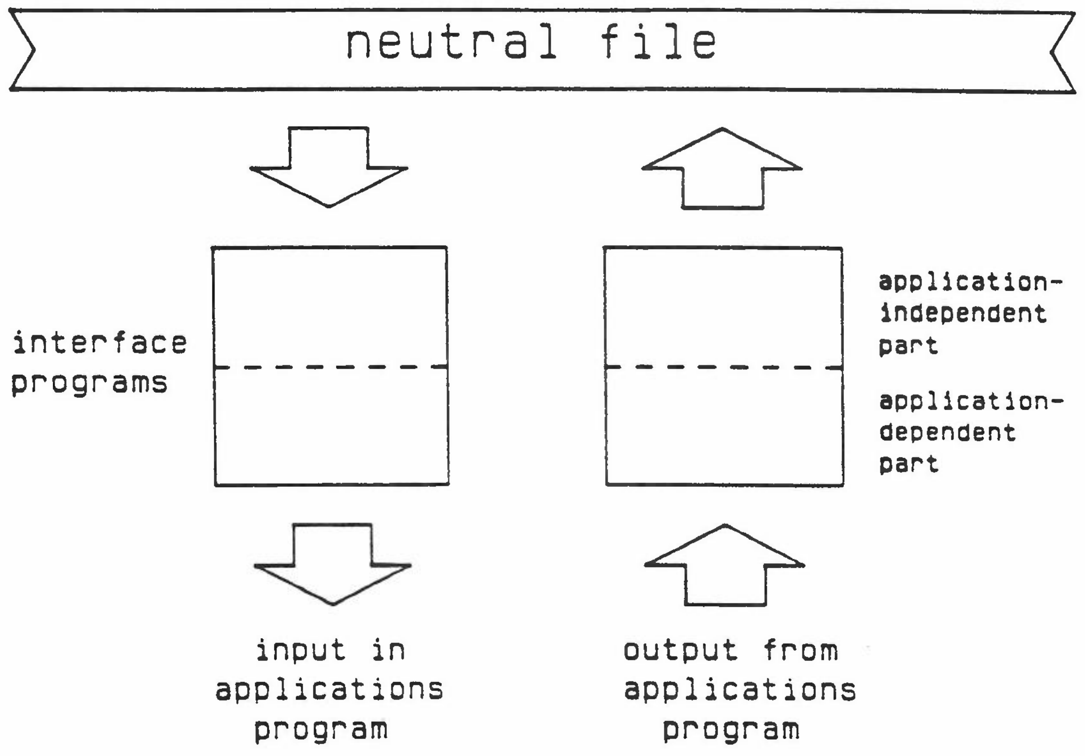

neutral file
interface programs
application-independent part
application-dependent part
input in applications program
output from applications program

Fig. 4. Interaction Between Neutral File and Applications.

The task of the CAD*I project is the definition of the neutral file as well as the development of the interfaces .

A.4.3 Internal Organisation of the File

The neutral file for product analysis data is structured as shown below:

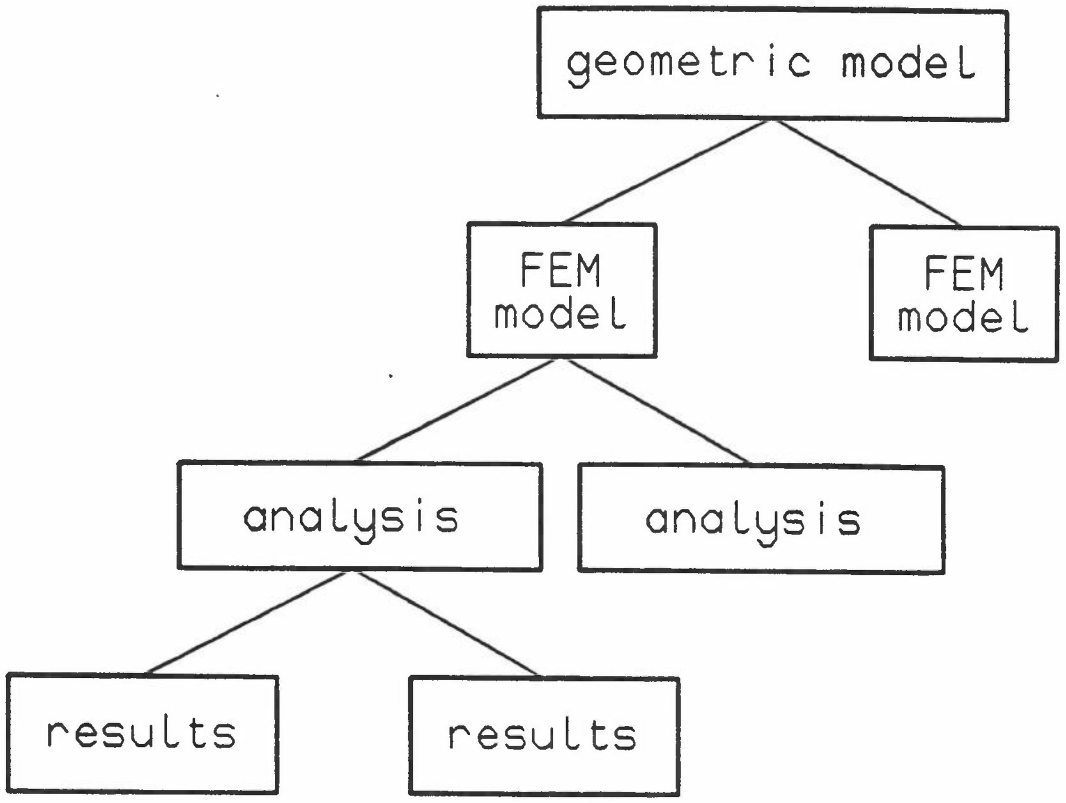


Fig. 5. Structure of neutral file.

This can be expressed in the neutral file by open/close constructs, which are not designed to be nested, but rather come one after another, each referring to the next higher level. The relevant block of information is always situated between the OPEN and the CLOSE statement. These keywords are detailed in section B.2.

A.5 External Envelope Concept

In the CAD*I environment files have to be transferred from one location to another. Often several files containing different types of information will be transmitted together. Then the problem occurs that the reading processor will get not only those files which it can interpret, but also files which were produced for being read by another processor.

In order to enable all processors at least to skip files which are unknown to them, an envelope concept is defined for all files in the whole CAD*I project. Each file gets a header and a trailer which indicates the area in which a processor is working. One or several files (each with its own header and trailer) form a so called metafile. That is the package which is actually transmitted. It also has a header and a trailer indicating the area where CAD*I files are stored. The header is dependent on the type of neutral file. The general construction is:

```
metafile header
header  of file 1
  .
  .   (content of file 1)
  .
trailer of file 1
header  of file 2
  .
  .   (content of file 2)
  .
trailer of file 2
header  of file 3
  .
  .   (content of file 3)
  .
trailer of file 3
metafile trailer
```

This scheme is also used if there is only one file to be transferred. Moreover, the concept allows different kinds of files, letters and even comments to be combined in one file. The header is dependent on the type of file. For the analysis data only the metafile header and a header for the neutral file is used. They are:

metafile header	:	CAD*I_FORMAT_BEGIN_19851011
metafile trailer	:	CAD*I_FORMAT_END___19851011
neutral file header	:	CAD*I_FORMAT_BEGIN_19881031
neutral file trailer	:	CAD*I_FORMAT_END___19881031

B Reference Section.

B.1 Syntax Definition

At the lowest level the neutral file is considered a stream of characters. The analysis of its actual contents takes place in two phases. The first phase (the lexical analysis) has as input the character stream and combines the characters into tokens, also it removes blanks and comments where required. The second phase (syntactical analysis) has as input the token stream produced by the lexical analyser and builds the syntax tree for the complete file, and in doing so it also drives the checking and interpretation of the file contents (semantic analysis). The basic character set is discussed in section B.1.1, the lexical analysis in section B.1.2 and syntactical analysis is the subject of section B.1.3.

B.1.1 The Alphabet

The list of possible characters in the file is called the alphabet. As the file is byte-oriented, the possible byte codes consist of the numbers 0 to 127. In this document these numbers are represented by graphical symbols according to the following (customary) mapping. Byte codes not occurring in this list (i.e. the numbers 0-31 and 127) are considered illegal in the neutral file and are not part of the alphabet. The graphical representation used in this document is given in the table.

32- 39	(space)	!	"	#	$	%	&	'
40- 47	(	)	*	+	,	-	.	/
48- 55	0	1	2	3	4	5	6	7
56- 63	8	9	:	;	<	=	>	?
64- 71	@	A	B	C	D	E	F	G
72- 79	H	I	J	K	L	M	N	O
80- 87	P	Q	R	S	T	U	V	W
88- 95	X	Y	Z	[	\	]	^	_
96-103	%	a	b	c	d	e	f	g
104-111	h	i	j	k	l	m	n	o
112-119	p	q	r	s	t	u	v	w
120-126	x	y	z	{	|	}	~	

B.1.2 Token Definitions

The first scanning of the file combines, where possible, separate characters of the alphabet into tokens (examples of tokens are identifiers and numbers). All legal tokens that can occur in the file form an infinite set. The elements of this set are defined using the notation of regular expressions. The rules to go from the regular expression to the associated sets are as follows:

If a is a character of the alphabet, then a used in a regular expression stands for the set {a}.

If R and S are regular expressions denoting the sets LR and LS, respectively, then R | S denotes the set LR U LS.

RS denotes the set of all strings from LR concatenated with all strings from LS.

R* denotes the set formed by taking elements from R zero or more times. For instance, if R consists of the set {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} then the empty string and the strings 88, 8878 are all elements of R*.

R+ denotes the set formed by taking elements from R one or more times. For instance, if R consists of the set {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} then the strings 88, 8878 are elements of R+.

Finally, [R] denotes the set of strings that are in R together with the empty string, i.e. it indicates that optionally one element of R is to be taken.

In the following list of expressions the curly braces {} are used as metasymbols. They are used to describe features of the language that are difficult or cumbersome to capture in formal language. The regular expressions that define the tokens in the neutral file are:

```
lower = a | b | ... | z | @ | [ | \ | ] | ^ | ' | { | } | ~
      (and the vertical bar '|' character itself)

upper = A | B | ... | Z | _

digit = 0 | 1 | ... | 9

space = (blank character)

special = { remaining characters in alphabet }

alphanum = lower | upper | digit

alphabet = lower | upper | digit |  special | space

non-q-char = (any character from the alphabet except the quote ' )

non-dq-char = (any character from the alphabet except the double
      quote " )

keyword = upper [ upper | digit ]*

name = [ alphanum ]+

entityname = # name

entityreference = entityname

sign = + | -

integer = [ sign ] digit+

real = [ sign ] digit+ . digit* [ E [ sign ] digit+ ]

string = ' [ non-q-char | ' ' ]* '

enumkeyword = upper [ upper | digit ]*

user-defined-name = " [ non-dq-character | " " ]* "

comment = ( * (any sequence of characters not containing *) )  * )
```

Additionally the following lexical conventions apply:

Spaces (blanks) separate tokens. Consecutive spaces can always be collapsed into one. The only exception is when spaces occur in strings, they are treated as any other character in a string.

Newlines are ignored altogether.

Comments can appear anywhere except within tokens. They are ignored except that they act as a token separator.

B.1.3 BNF Description of the File

The BNF rules give a formal description of the syntax of the file. The terminals of the language are the tokens as recognised by the lexical analyser. For convenience in running the parser an extra production (the first) is added and the start symbol of the grammar is therefore <SYSTEM GOAL SYMBOL>. Production no. 24 provides for the case of a missing or default parameter, as for instance occurs in the following statement in the neutral file:

```
NODE(#99: 15, 4.4, 5.5, 6.6, , #FR001);
```

The lexical analyser is assumed to return the special token eof when it encounters the end of the neutral file. The productions are:

```
 1 <SYSTEM GOAL SYMBOL> ::= <neutralfile>
 2 <neutralfile> ::= <file> eof
 3 <file> ::= <statement list> ;
 4 <statement list> ::= <statement>
 5      / <statement list> ; <statement>
 6 <statement> ::= <definition>
 7               / <property>
 8 <definition> ::= keyword ( entityname : <parameter list> )
 9 <property> ::= keyword ( <parameter list> )
10 <parameter list> ::= <parameter>
11                    / <parameter list>,<parameter>
16 <parameter> ::= real
17               / string
18               / integer
19               / entityreference
20               / enumkeyword
21               / user-defined-name
22               / <embedded entity>
23               / ( <parameter list> )
24               /
25 <embedded entity> ::= keyword ( <parameter list> )
```

B.1.4 Description of EXPRESS for FE specification

The main part of this report is a description of all the keywords with their parameters. Each keyword in section B.3 to B.5 is described in the same way: a header line (printed in bold) which gives a quick overview of the keyword and its parameters, followed by an Express-like definition of the entity type for giving a formal definition of its attributes, a description of the entity being defined and the meaning of the entity and the parameters. This is finally followed by an example of how an instance of such an entity would look in the neutral file.

Express is a language being developed by the International Standards Organisation. A full description of the language can be found in ISO document ISO/TC184/SC4/WG1 and here will only be given a summary of the aspects relevant for understanding the information in Part B. With the aim of better serving the purposes of the CAD*I project the language has been slightly modified as compared with the original version.

B.1.4.1 Entities and Attributes

An Express entity type definition consists of a name, a list of attributes and optionally a where clause. The name, occurring after the keyword ENTITY, allows references to the entity to be made by other entities in the Express model and always corresponds with the name as used in the neutral file. The list of attributes in the Express definition corresponds with the parameters in the neutral file. The where clause puts restrictions on the possible values of the attributes. Comments are allowed within the comment delimeters '(*' and '*)'.

The attribute list immediately follows the entity name. For each attribute a name and a type is given, separated by a colon ':'. Where consecutive attributes have the same type their names can be combined into a list of names with the names separated by comma's. A list of possible attributes types, together with the way they are represented on the neutral file, follows:

B.1.4.2 Base Types

An integer type means that the attribute is an integer and is represented on the neutral file as such.

A real type means that the attribute is a real number and is represented on the neutral file as a real.

A string type means that the attribute is a string and is represented on the neutral file as a string.

B.1.4.3 Entity references

An entity type name means that the attribute is a reference to an entity of that type. On the neutral file it is represented either by an explicit reference to an entity or an embedded entity of the correct type.

A select means that the attribute is a reference to an entity the type of which cannot be determined in advance. Instead, the possible types that can be referenced are listed explicitly as parameters of the SELECT. On the neutral file such a parameter is either an explicit reference or an embedded entity.

B.1.4.4 Enumeration

An enumeration means that the attribute has a relatively small number of values which are listed explicitly as the parameters of the enumeration. On the neutral file it is represented by the name of the selected value.

B.1.4.5 Unique and Optional

The attribute types can be modified by the addition of the words unique or optional. These have the following meaning:

If an attribute is labelled unique this means that the value of instances of the attribute should be unique throughout the model.

An optional attribute does not need to be present in an instance of the entity on the neutral file. If the attribute is not present then its place on the neutral file is left empty. The separators for the attribute are not deleted so an omitted attribute is characterised by two consecutive comma's.

B.1.4.6 Aggregates

It is possible for the type of an attribute to be composite in the sense that it is a number of values instead of a single value. Three aggregate types exist: array, list and set. An array is used when the values can only by addressed by their specific position in the array. An array attribute can be thought of as providing 'slots' or placeholders for the values and so an array has a field length which can be defined in the Express model itself. A list is used when the collection of values is ordered but not explicitly numbered. A set is used when the collection of values is unordered and no duplication of values is allowed, which corresponds with the mathematical concept of a set.

[illegible]

B.1.4.7 Other definition constructs

In addition to entity type definitions the following Express constructs can be used.

TYPE. A type definition is a convenient way of writing select clauses outside the main body of the entity. The same select is often shared between entities as the type of one of their attributes and a type definition provides the facility for sharing selects between several entity definitions.

PROPERTY. A property definition can be used to add information to an already existing entity. The express syntax consists of the word PROPERTY, an attribute list, followed by the keyword OF, and a reference (possibly a select). The reference is used for stating which entity the property attributes belong to.

LINK. A link can be used to mention facts that do not properly belong to a single entity but to a combination of entities. A link can itself have attributes.

B.1.4.8 Constraints

The where clause puts restrictions on the values of the attributes. The where clauses as used in part B have a relatively simple syntax and, with one exception, the interpretation of their meaning should be obvious. This exception is the use of the word 'NULL'. If an attribute A is optional it is possible to formulate a constraint involving a test on the occurrence of attribute A in the actual entity using the statement 'A=NULL' (which returns a logical value).

It must be kept in mind that it is not intended to capture all possible restrictions on all attribute values of the entities. This would be very difficult to do as there is a gradual transition from direct constraints on the file (example: NODAL, B.4.4, where type-code should be between 1 and 4), constraints that can be formulated in a simple manner as only one attribute is involved (for example Poisson's ratio less than 1/2) to complicated constraints that can only be checked by a full perusal of the model (example: a model that is not sufficiently restrained so that its stiffness matrix is singular). As a general rule only constraints that exist for the purposes of the file itself are mentioned and simple other constraints have been added where this seemed appropriate.

The implication of this is that not every neutral file which is legal under the definitions of this report corresponds with a realistic analysis problem: it is feasible (and, in fact, quite easy) to design a file which fully complies with the formats and constraints on the values in the neutral file, but which nevertheless does not represent a model which is realistic in any sense.

B.2 Levels of the Neutral File

The neutral file is divided into four levels, corresponding to stages in the design cycle of a product. The statements belonging to a particular level are enclosed between OPEN and CLOSE keywords in the physical file to separate them from the rest. This means that a neutral file may contain many different types of information in an ordered block structure, rather than an unorganised heap.

The following keywords are provided for these constructs:

FEM level 1 (geometrical structure)

this is provided by CAD (overlapping area, fig. 3.) and may begin with OPENWORLD. This world has a name, which can be referred by other levels. The appropriate keywords and statements are given in the CAD*I specification of a neutral file for CAD geometry [31].

FEM level 2 (FEM model):

OPENMODEL (model_name:world_name,text_description);

CLOSEMODEL (model_name);

model_name can be referenced by the following levels,

world_name is the name of the geometrical structure described between OPENWORLD and CLOSEWORLD,it can be omitted (e.g. because there is no product definition data)

text_description is a string constant.

The colon (:) indicates that in this statement a new entity is created (defined).

FEM level 3 (analysis):

OPENANA (analysis_name:model_name,text_description);

CLOSEANA (analysis_name);

model_name was defined in OPENMODEL, analysis_name may be referenced by the result sets. The analysis is not placed inside the model description, but in an unnested way after the CLOSEMODEL statement.

FEM level 4 (FE results):

OPENFERES (set_name:analysis_name,text_description);

CLOSEFERES (set_name);

set_name is a name for this special set of results (e.g. displacements or stresses of one load case), the text may give further information about this set (e.g. 7th eigenvector, frequency = 254.3 Hz).

The structure explained here is applied to the example given in chapter A.3.

B.3 Keywords for FE Model Description

B.3.1 OPENMODEL (model_name:world_name,'TEXT_DESCRIPTION');

```
ENTITY openmodel;
   world_name           : OPTIONAL openworld;
   TEXT                 : OPTIONAL string;
END_ENTITY;
```

model_name can be referenced by the following levels,

world_name is the name of the geometrical structure described between OPEN_WORLD and CLOSE_WORLD; if no product definition data exist, world_name may be omitted.

TEXT_DESCRIPTION is an optional string constant.

Example:

```
OPENMODEL (#testmodel: ,'This model has no world');
```

B.3.2 UNITS (modelname, LE, MA, TI,TE, TE_OFFSET, CU, LI, ANGLE);

```
PROPERTY units;
   LE,MA,TI,TE          : OPTIONAL real;
   TE_OFFSET,CU,LI      : OPTIONAL real;
   ANGLE                : OPTIONAL real;
OF
   MODELNAME            : openmodel
END_PROPERTY;
```

modelname, name of the FEM model
LE, length units per meter
length in m = length in local units / LE
MA, mass units per kilogram
TI, time units per second
TE, temperature units per kelvin
TE_OFFSET, offset of zero point in temperature
temp. in kelvin = temp. in local units / TE + offset
CU, current units per ampere
LI, light units per candela
ANGLE, units per full circle

default value for LE, MA, TI, TE, CU, LI is 1
default value for TE_OFFSET is 0, for ANGLE 6.2831853... (2 pi).

Examples:

UNITS (#testmodel,100.,0.01,,,273.15,,,360.);

assigns a unit system with centimeters, tenth of a ton, seconds and degrees Celsius to structure #testmodel. The angle unit is a degree.

B.3.3 COORD (newcosys:oldcosys,C1,C2,C3,PHI,THE,PSI,MIRROR, TYPE_CODE);

```
ENTITY coord;
   oldcosys              : OPTIONAL coord;
   C1, C2, C3            : real;
   PHI,THE,PSI           : real;
   MIRROR                : OPTIONAL integer;
   TYPE                  : ENUMERATION OF ( RECT,CYLN,SPHR );
WHERE
   MIRROR >= 0 AND MIRROR <= 3;
END_ENTITY;
```

newcosys is the name of the coordinate system being defined. oldcosys is the name of the referred (existing) coordinate system; if omitted reference is made to the global (cartesian) coordinate system

C1,C2,C3 are the coordinates of the origin of the new system relative to the existing coordinate system 'oldcosys'. They may be x,y,z values, r,theta,z or r,theta,psi depending on whether 'oldcosys' is a rectangular, cylindrical or spherical coordinate system.

PHI,THE,PSI are the Euler angles describing the orientation of the new coordinate system relative to the old system. These angles always refer to an imaginary rectangular cosys thus fixing the orientation of any type of cosys. Positive angles are clockwise when looking in a positive direction along the rotation axis.

To obtain the new system the old system is rotated first about the x-axis by an angle PHI producing new y and z axes. Then this intermediate coordinate system is rotated by an angle THE about its intermediate y axis producing new x and z axes. The last rotation is an angle PSI around this new z axis (which is the final one) producing the final x and y axes. This transformation, from the XYZ system to the new coordinate system xyz, is given by the matrix below.

$$\begin{Bmatrix} x \\ y \\ z \end{Bmatrix} = [\ \text{matrix}\ ] * \begin{Bmatrix} X \\ Y \\ Z \end{Bmatrix}$$

The elements of the matrix are:

$$\begin{array}{lll}
\cos THE \cos PSI & \cos PHI \sin PSI + & \sin PHI \sin PSI - \\
 & \sin PHI \sin THE \cos PSI & \cos PHI \sin THE \cos PSI \\
 & & \\
-\cos THE \sin PSI & \cos PHI \cos PSI - & \sin PHI \cos PSI + \\
 & \sin PHI \sin THE \sin PSI & \cos PHI \sin THE \sin PSI \\
 & & \\
\sin THE & -\sin PHI \cos THE & \cos PHI \cos THE
\end{array}$$

MIRROR is an integer number denoting which axis of the newcosys shall be inverted (mirrored). If omitted or set to zero then no mirroring is performed.

TYPE_CODE can be RECT,CYLN or SPHR to specify the type of the new coordinate system (rectangular, cylindrical, spherical respectively).

Example:

```
COORD (#co7: #co6, 0.,1.,0., 0.,-90.,0., 0,RECT);
      defines a new coordinate system #co7, which relatively to the old
      system #co6 is shifted in y-direction and rotated around the y-
      axis.
```

B.3.4 FREEDOM (name:DOFS,cosys);

```
ENTITY freedom;
   DOFS                 : string;
   cosys                : OPTIONAL coord;
WHERE
   (* DOFS has all sorts of constraints in it *)
END_ENTITY;
```

Defines for three translations and three rotations which are fixed.

DOFS is a string constant of up to 6 numeric characters (1...6). Free degrees-of-freedom are not mentioned.

Examples are:

'123456'	fix all 6 degrees of freedom
'456'	fix rotations
'3'	fix z-direction
''	all degrees of freedom are free

Degrees of freedom which will be condensed or will get prescribed displacements must be declared free (this is the default value for not mentioned degrees of freedom).

cosys is the name of a coordinate system

See note (1).

Example:

```
FREEDOM (#F35: '135',);
       fixes directions x,z,phi-y in global coordinate system.
```

B.3.5 NODE (name:NODE_NUMBER,X,Y,Z,cosys,freedom);

```
ENTITY node;
   NODE_NUMBER            : UNIQUE integer;
   X, Y, Z                : real;
   cosys                  : OPTIONAL coord;
   freedom                : freedom;
WHERE
   NODE_NUMBER > 0;
END_ENTITY;
```

NODE_NUMBER is the node number in the FEM programs,

X,Y,Z are coordinates of type real,

cosys is the name of a coordinate system; if omitted X,Y,Z are given in the model coordinate system, if no model coordinate system is specified, it defaults to a global cartesian system.

freedom is the name of a set of degrees-of-freedom.

See note (1).

Example:

```
NODE (#N117: 117,3.15,-2.52,-0.05,#Col,#F35);
       defines node 117 with coordinates given in coordinate system #Col
       and degrees of freedom given by #F35.
```

B.3.6 ISO (name:E,NUE,RHO);

```
ENTITY iso;
   E                       : real;
   NUE                     : real;
   RHO                     : real;
END_ENTITY;
```

Defines a minimal set of properties of an isotropic material:

E	- elastic modulus
NUE	- Poisson's ratio
RHO	- density

Example:

```
ISO (#isl:2.1E11, 0.3, 7850.);
```

B.3.7 ISOFULL (name:E,NUE,RHO,AL,BE,EPS,H,TREF,K,CP);

```
ENTITY isofull;
   E                      : real;
   NUE                    : real;
   RHO                    : real;
   AL                     : OPTIONAL real;
   BE                     : OPTIONAL real;
   EPS                    : OPTIONAL real;
   H                      : OPTIONAL real;
   TREF                   : OPTIONAL real;
   K                      : OPTIONAL real;
   CP                     : OPTIONAL real;
END_ENTITY;
```

Fully defines the properties of an isotropic material:

E	- elastic modulus
NUE	- Poisson's ratio
RHO	- density
AL	- proportionality factor for damping (stiffness contribution)
BE	- proportionality factor for damping (mass contribution) [C] - AL*[K] + BE*[M]
EPS	- critical damping ratio
H	- thermal expansion coefficient
TREF	- thermal expansion reference temperature
K	- thermal conductivity coefficient
CP	- heat capacity

Example:

```
ISOFULL (#if2:2.1E11, 0.3, 7850.0,o.5,0.5,1.3,12E-5,300.,100.,3700.);
       defines the properties of an isotropic material.
```

B.3.8 ANISO2D (name:EM,RHO,AL,BE,EPS,HM,TREF,KM,CP,cosys);

```
ENTITY aniso2d;
   EM                     : ARRAY [1 : 6] OF real;
   RHO                    : real;
   AL                     : OPTIONAL real;
   BE                     : OPTIONAL real;
   EPS                    : OPTIONAL real;
   HM                     : OPTIONAL ARRAY [1 : 3] OF real;
   TREF                   : OPTIONAL real;
   KM                     : OPTIONAL ARRAY [1 : 3] OF real;
   CP                     : OPTIONAL real;
   cosys                  : OPTIONAL coord
END_ENTITY;
```

Defines the properties of a 2D anisotropic material:

NUE	- Poisson's ratio
RHO	- density
AL	- proportionality factor for damping (stiffness contribution)
BE	- proportionality factor for damping (mass contribution) [C] - AL*[K] + BE*[M]
EPS	- critical damping ratio
TREF	- thermal expansion reference temperature
CP	- heat capacity
cosys	- reference to a coordinate system, if this is omitted the default is the local element coordinate system
EM (3x3)	- 2D elasticity matrix (6 values)
HM (2x2)	- 2D thermal expansion matrix (3 values)
KM (2x2)	- 2D thermal conductivity matrix (3 values)

The matrices are given as lists of values row by row beginning in the main diagonal:

```
A11,A12,A13,...A1n,
    A22,A23,...A2n,    with n - 2,3 or 6
        ..........,
               Ann
```

The list of values is regarded as one parameter and is therefore included in parentheses.

Example:

```
ANISO2D (#an2:(2.1E10,2.2E10,2.3E10,2.4E10,2.5E10,2.6E10),
              7000.,1.E-6,0.44,,(4.1E3,4.2E3,4.3E3),20.,
              (5.1E-2,5.2E-2,5.3E-2),0.341,,);
```

lists the properties of an anisotropic material for 2D elements in the global coordinate system.

B.3.9 ANISO3D (name:EMM,RHO,AL,BE,EPS,HMM,TREF,KMM,CP,cosys);

```
ENTITY aniso3d;
   EMM                  : ARRAY [1 : 21] OF real;
   RHO                  : real;
   AL                   : OPTIONAL real;
   BE                   : OPTIONAL real;
   EPS                  : OPTIONAL real;
   HMM                  : OPTIONAL ARRAY [1 : 6] OF real;
   TREF                 : OPTIONAL real;
   KMM                  : OPTIONAL ARRAY [1 : 6] OF real;
   CP                   : OPTIONAL real;
   cosys                : OPTIONAL coord
END_ENTITY;
```

Defines the properties of a 3D anisotropic material:

RHO	= density
AL	= proportionality factor for damping (stiffness contribution)
BE	= proportionality factor for damping (mass contribution) [C] = AL*[K] + BE*[M]
EPS	= critical damping ratio
TREF	= thermal expansion reference temperature
CP	= heat capacity
cosys	= reference to a coordinate system, if this is omitted the default is the local element coordinate system
EMM (6x6)	= 3D elasticity matrix (21 values)
HMM (3x3)	= 3D thermal expansion matrix (6 values)
KMM (3x3)	= 3D thermal conductivity matrix (6 values)

The matrices are given as lists of values row by row beginning in the main diagonal:

```
A11,A12,A13,...A1n,
    A22,A23,...A2n,    with n = 2,3 or 6
        ..........,
               Ann
```

The list of values is regarded as one parameter and is therefore included in parentheses.

Example:

```
ANISO3D(#an3:(2.1E10,2.2E10,2.3E10,2.4E10,2.5E10,2.6E10,
        2.7E10,2.7E10,2.7E10,2.7E10,2.7E10,2.7E10,2.7E10,
        2.7E10,2.7E10,2.7E10,2.7E10,2.7E10,2.7E10,2.7E10,
        2.7E10,),7000.,1.E6,0.44,,(4.1E3,4.2E3,4.3E3,
        4.1E3,4.2E3,4.3E3),20.,
        (5.1E-2,5.2E-2,5.3E-2,5.1E-2,5.2E-2,5.3E-2)
        ,0.341,,);
```

defines the properties of an anisotropic material for 3D elements in the global coordinate system.

B.3.10 MATERIAL (name:'TEXT',mat_subkw);

```
ENTITY material;
   TEXT                 : OPTIONAL string;
   mat_subkw            : SELECT (iso,isofull,aniso2d,
                                  aniso3d);
END_ENTITY;
```

Defines the material properties associated with an element. The material may be designated isotropic or anisotropic depending on the keyword that is referenced.

TEXT is an optional string, usually describing the material.

Example:

```
MATERIAL (#matl:'Structural Steel',#iso4);
```

defines a material, matl, the properties of which are given in #iso4.

B.3.11 PBAR (name:A);

```
ENTITY pbar;
   A                       : real;
WHERE
   A > 0.
END_ENTITY;
```

Defines the properties of a bar element:

A	cross-sectional area

This may be referenced by elements of type BAR.

Example:

```
PBAR (#bar12:12.4E-4);
```

B.3.12 BEAMG (name:A,AIYY,AIZZ,AIYZ,AIP,SY,SZ,DY,DZ);

```
ENTITY beamg;
   A                     : OPTIONAL real;
   AIYY                  : OPTIONAL real;
   AIZZ                  : OPTIONAL real;
   AIYZ                  : OPTIONAL real;
   AIP                   : OPTIONAL real;
   SY                    : OPTIONAL real;
   SZ                    : OPTIONAL real;
   DY                    : real;
   DZ                    : real;
WHERE
   A > 0. ;
   AIYY > 0. ;
   AIZZ > 0. ;
   AIYZ > 0. ;
   AIP > 0. ;
   SY > 0. ;
   SZ > 0. ;
   NOT ( (A=NULL) AND (AIYY=NULL) AND (AIZZ=NULL)
   AND (AIYZ=NULL) AND (SY=NULL) AND (SZ=NULL) );
END_ENTITY;
```

Defines the properties of a general beam element:

A	cross-sectional area; if omitted the beam has no resistance against tension and compression
AIYY,AIZZ,AIYZ	second moments of area; if omitted the beam has no resistance against bending
AIP	torsional constant; if omitted the beam has no resistance against torsion
SY,SZ	shear coefficients (cross-sectional area divided by effective shear area); if omitted the beam has no resistance against shear
DY,DZ	local coordinates defining the point at which stress is to be computated.

This may be referenced by elements of type BEAM2.

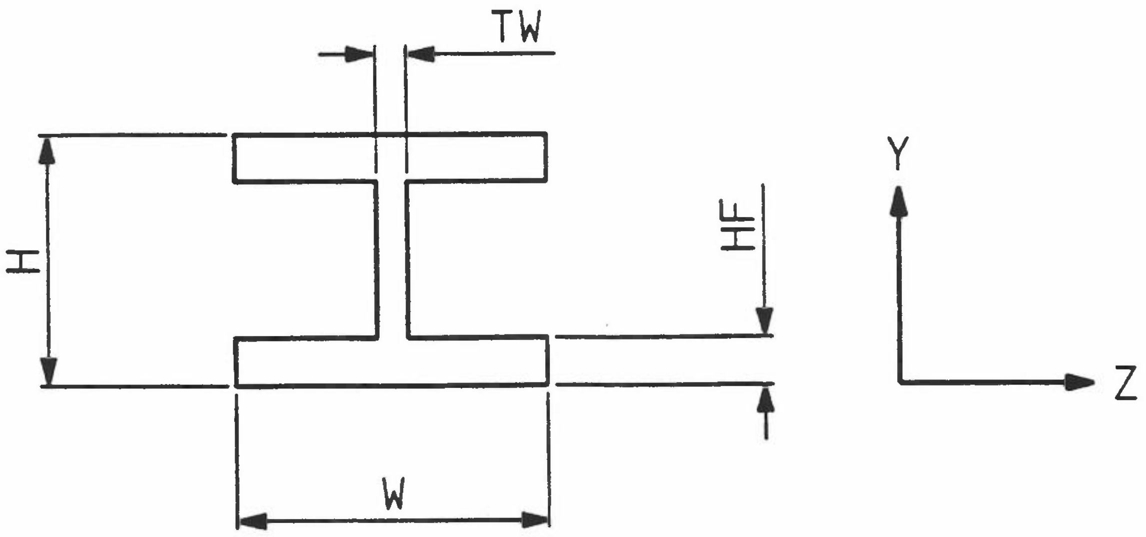
Example:

```
BEAMG (#b1:123.,342.,774.3,125.0,545.3,1.5,1.6,15.2,17.6);
```

B.3.13 BEAMI (name:H,W,HF,TW);

```
ENTITY beami;
   H                        : real;
   W                        : real;
   HF                       : real;
   TW                       : real;
WHERE
   HF > 0.;
   H  > (2 * HF);
   TW > 0.;
   W  > TW;
END_ENTITY;
```

Defines the properties of an I section beam:



H	height of a profiled beam (size in local y'-direction)
W	width of a profiled beam (size in local z'-direction)
HF	height of flange (measured in local y'-direction)
TW	thickness of web (measured in local z'-direction)

The neutral axis is assumed to lie along the x-axis. This property may be referenced by elements of type BEAM2.

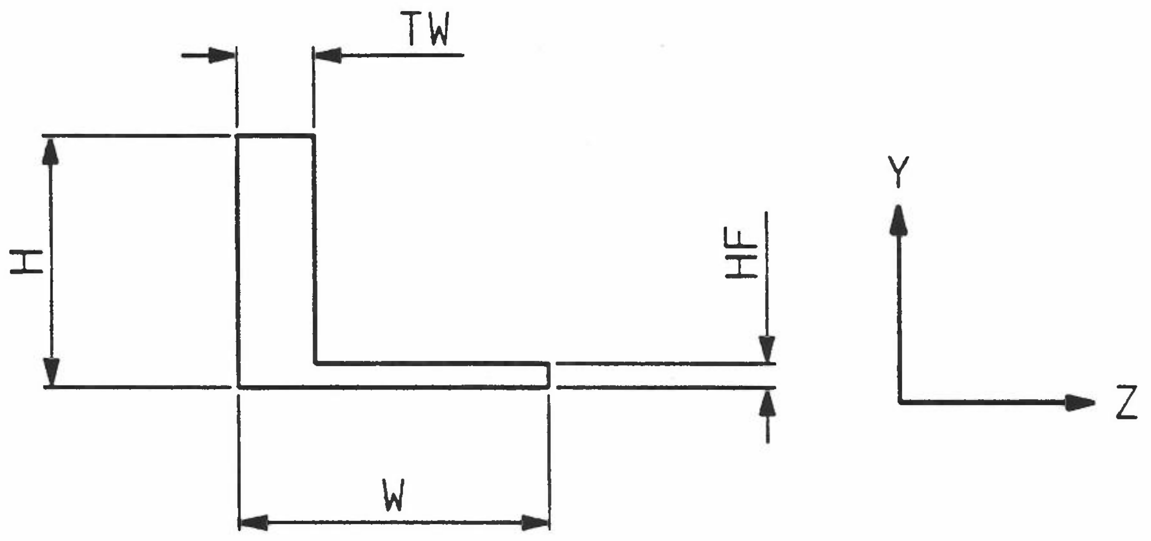
Example:

```
BEAMI (#bI5:24.5,13.7,1.2,4.5);
```

B.3.14 BEAML (name:H,W,HF,TW);

```
ENTITY beaml;
   H                        : real;
   W                        : real;
   HF                       : real;
   TW                       : real;
WHERE
   HF > 0.;
   H  > HF;
   TW > 0.;
   W  > TW;
END_ENTITY;
```

Defines the properties of an L section beam:



H	height of a profiled beam (size in local y'-direction)
W	width of a profiled beam (size in local z'-direction)
HF	height of flange (measured in local y'-direction)
TW	thickness of web (measured in local z'-direction)

The neutral axis is assumed to lie along the x-axis. This property may be referenced by elements of type BEAM2.

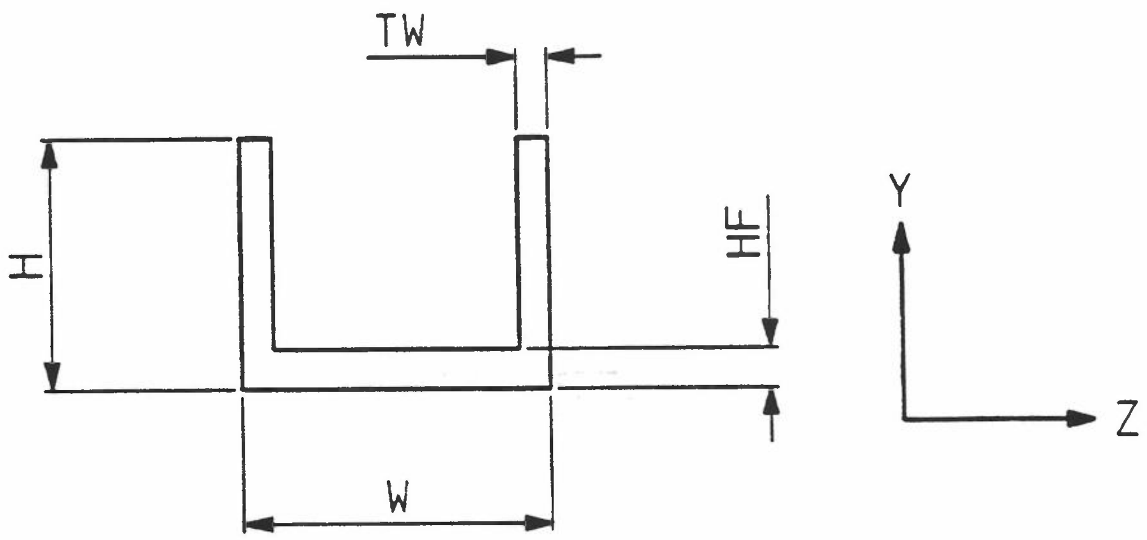
Example:

```
BEAML (#bL7:24.5,13.7,1.2,4.5);
```

B.3.15 BEAMU (name:H,W,HF,TW);

```
ENTITY beamu;
   H                       : real;
   W                       : real;
   HF                      : real;
   TW                      : real;
WHERE
   HF > 0.;
   H  > HF;
   TW > 0.;
   W  > (2 * TW);
END_ENTITY;
```

Defines the properties of a U section beam:



H	height of a profiled beam (size in local y'-direction)
W	width of a profiled beam (size in local z'-direction)
HF	height of flange (measured in local y'-direction)
TW	thickness of web (measured in local z'-direction)

The neutral axis is assumed to lie along the x-axis. This property may be referenced by elements of type BEAM2.

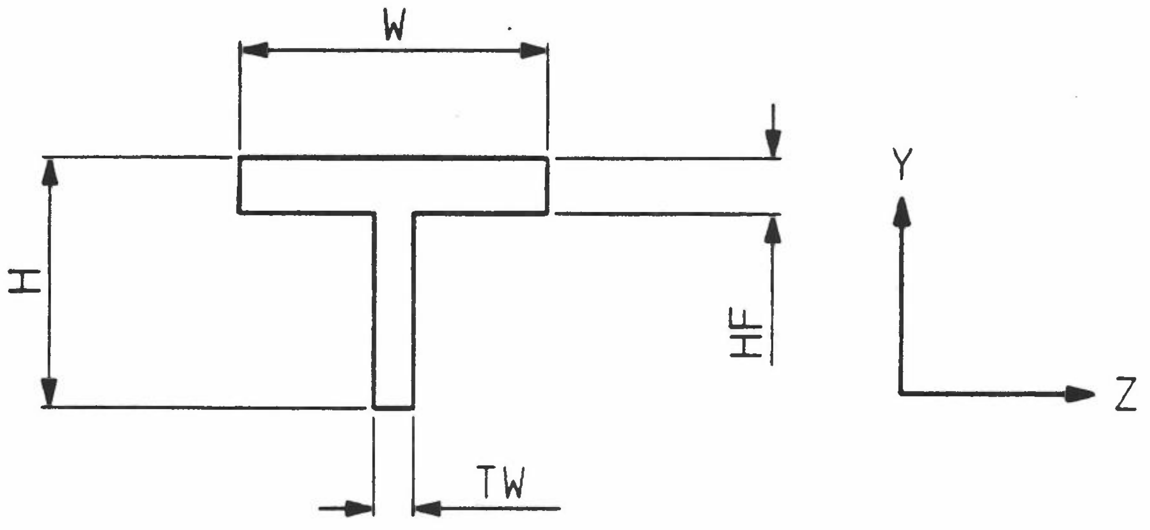
Example:

```
BEAMU (#bU1:24.5,13.7,1.2,4.5);
```

B.3.16 BEAMT (name:H,W,HF,TW);

```
ENTITY beamt;
   H                          : real;
   W                          : real;
   HF                         : real;
   TW                         : real;
WHERE
   HF > 0.;
   H  > HF;
   TW > 0.;
   W  > TW;
END_ENTITY;
```

Defines the properties of an T section beam:



H	height of a profiled beam (size in local y'-direction)
W	width of a profiled beam (size in local z'-direction)
HF	height of flange (measured in local y'-direction)
TW	thickness of web (measured in local z'-direction)

The neutral axis is assumed to lie along the x-axis. This property may be referenced by elements of type BEAM2.

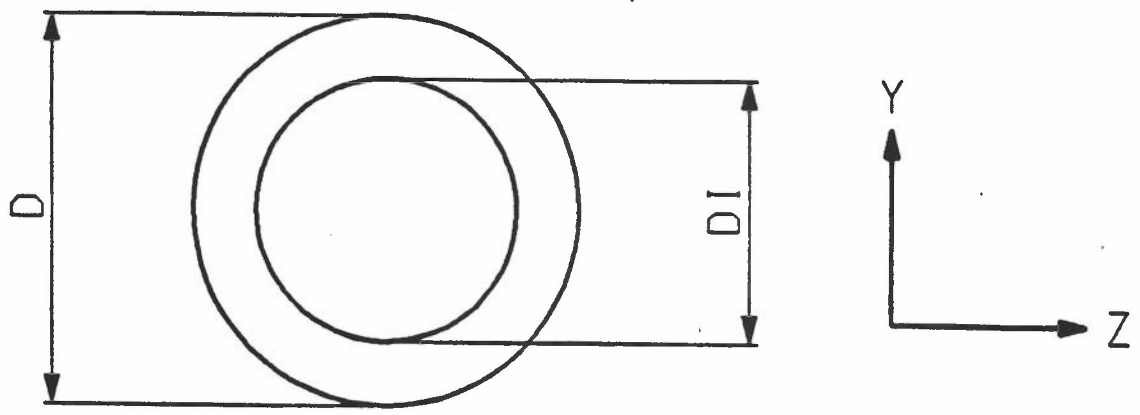
Example:

```
BEAMT (#bT91:24.5,13.7,1.2,4.5);
```

B.3.17 BEAMO (name:D,DI);

```
ENTITY beamo;
   D                       : real;
   DI                      : OPTIONAL real;
WHERE
   D > 0.;
   D > DI;
END_ENTITY;
```

Defines the properties of an O section beam:



D	outer diameter of a round beam
DI	inner diameter of a round beam (may be omitted or set zero in case of solid beam)

The neutral axis is assumed to lie along the x-axis. This property may be referenced by elements of type BEAM2.

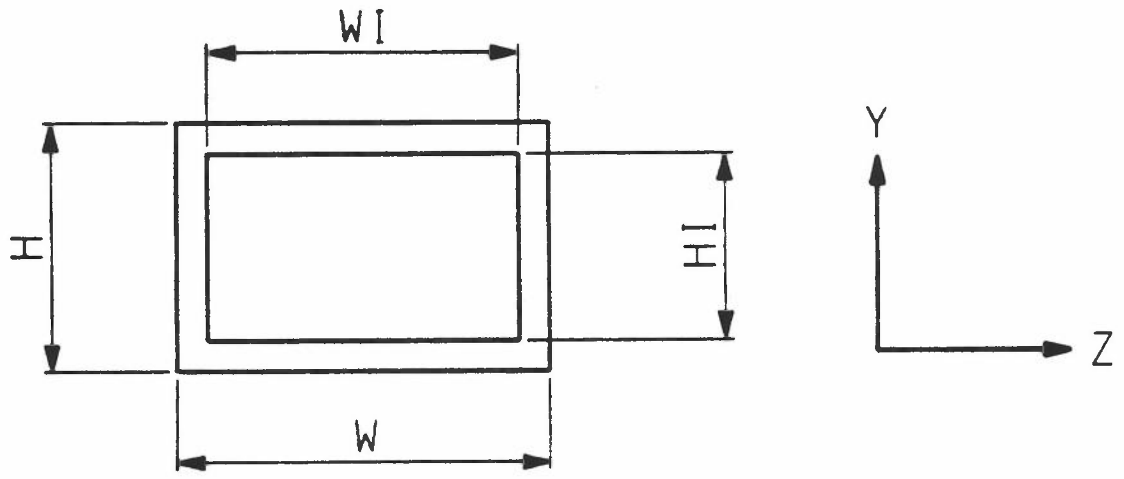
Example:

```
BEAMO (#b065:99.9,43.5);
```

B.3.18 BEAMR (name:H,W,HI,WI);

```
ENTITY beamr;
   H                    : real;
   W                    : real;
   HI                   : OPTIONAL real;
   WI                   : OPTIONAL real;
WHERE
   HI > 0.;
   H  > HI;
   WI > 0.;
   W  > WI;
END_ENTITY;
```

Defines the properties of a rectangular section beam:



H	height of a profiled beam (size in local y'-direction)
W	width of a profiled beam (size in local z'-direction)
HI,WI	inner height, inner width of rectangular beams (may be omitted or set zero in case of solid beam)

The neutral axis is assumed to lie along the x-axis. This property may be referenced by elements of type BEAM2.

Example:

```
BEAMR (#br001:33.4,55.6,12.6,23.9);
```

B.3.19 STIFF (name:KX,KY,KZ,KXX,KYY,KZZ);

```
ENTITY stiff;
   KX                     : OPTIONAL real;
   KY                     : OPTIONAL real;
   KZ                     : OPTIONAL real;
   KXX                    : OPTIONAL real;
   KYY                    : OPTIONAL real;
   KZZ                    : OPTIONAL real;
END_ENTITY;
```

Defines the properties of a SPRING element:

KX,KY,KZ	translational stiffnesses, valid for global directions if auxiliary point is absent, valid for local directions if local coordinate system is given (by ORIENT)
KXX,KYY,KZZ	rotational stiffness around global or local axes (depends on whether an auxiliary point is given or not)

Example:

STIFF (#sti:9.1E6,7.1E6,5.1E6,0.0,8.1E6,0.0);
defines translational and rotational stiffnesses for a spring element.

B.3.20 DAMP (name:CX,CY,CZ,CXX,CYY,CZZ);

```
ENTITY damp;
   CX                    : OPTIONAL real;
   CY                    : OPTIONAL real;
   CZ                    : OPTIONAL real;
   CXX                   : OPTIONAL real;
   CYY                   : OPTIONAL real;
   CZZ                   : OPTIONAL real;
END_ENTITY;
```

Defines the properties of a DAMPER element:

CX,CY,CZ	translational damping coefficients, valid for global or local directions
CXX,CYY,CZZ	rotational damping coefficients around global or local axes

Example:

DAMP (#dmp: 1.5,2.4,0.0,12.9,0.0,11.1);
defines translational and rotational damping coefficients for a damper element.

B.3.21 PMASS (name:MX,MY,MZ,MXX,MYY,MZZ);

```
ENTITY pmass;
   MX                     : OPTIONAL real;
   MY                     : OPTIONAL real;
   MZ                     : OPTIONAL real;
   MXX                    : OPTIONAL real;
   MYY                    : OPTIONAL real;
   MZZ                    : OPTIONAL real;
END_ENTITY;
```

Defines the properties of a lumped mass element:

MX,MY,MZ	translational mass (usually equal for all 3 directions) in global or local coordinates.
MXX,MYY,MZZ	rotational moments of inertia around global or local axes

This property type may be referenced only by an element of type MASS.

Example:

```
PMASS (#pm85:2.4,3.1,82.4,2.7,45.3,44.9);
```

B.3.22 THICK (name:PURPOSE_CODE,T);

```
ENTITY thick;
   PURPOSE_CODE         : ENUMERATION OF ( MEMBRANE,PLANESTRAIN,
                                      PLATE,SHELL,AXISOLID,SHEAR);
   T                    : real;
WHERE
   T > 0.;
END_ENTITY;
```

Defines the properties of a 2D element:

T	thickness of a 2D element (constant over the whole element)
P	purpose code giving the type of 2D element MEMBRANE - membrane element having plane stress PLANESTRAIN - membrane element having plane strain SHELL - thin shell element PLATE - thin plate element AXISOLID - axisymmetric ring element SHEAR - a 2D element where transverse shear is important

This property may be referenced by TRI and QUAD type elements.

Example:

```
THICK (#thk4:MEMBRANE,1.0E-3);
       defines the thickness of a membrane element.
```

B.3.23 THICKV (name:PURPOSE_CODE,T1,T2,T3,T4);

```
ENTITY thickv;
   PURPOSE_CODE          : ENUMERATION OF ( MEMBRANE,PLANESTRAIN,
                                         PLATE,SHELL,AXISOLID,SHEAR);
   T1, T2, T3            : real;
   T4                    : OPTIONAL real;
WHERE
   T1 > 0.;
   T2 > 0.;
   T3 > 0.;
   T4 > 0.;
END_ENTITY;
```

Defines the properties of a 2D element:

T1,T2,T3,T4	thicknesses at the 1st,2nd,3rd,4th corner points of a 2D element (in case of triangles T4 is omitted)
P	purpose code giving the type of 2D element MEMBRANE - membrane element having plane stress PLANESTRAIN - membrane element having plane strain SHELL - thin shell element PLATE - thin plate element AXISOLID - axisymmetric ring element SHEAR - a 2D element where transverse shear is important

This property may be referenced by TRI and QUAD type elements.

Example:

```
THICKV (#thkv2:PLATE,1.0E-3,2.0E-3,3.0E-3,4.0E-3);
       defines the varying thickness of a plate element with four nodes.
```

B.3.24 PROPERTY (name:prop_subkw);

```
ENTITY property;
   prop_subkw           : SELECT ( pbar,beamg,beami,beaml,beamu,
                                   beamt,beamo,beamr,stiff,damp,
                                   pmass,thick,thickv ) ;
END_ENTITY;
```

This statement defines the element properties and gives them a name that can be referenced by the element statement. This keyword references a keyword containing the properties relevant to a particular type of element.

Example:

```
PROPERTY (#prop4:#thkv19);
```

B.3.25 OR1 (name:node);

```
ENTITY or1;
   a_node                : node;
END_ENTITY
```

Specifies a node, referenced by the ORIENT keyword, to define a direction.

Example:

```
OR1 (#or22:#n467);
```

B.3.26 OR3 (name:X,Y,Z);

```
ENTITY or3;
   X,Y,Z                 : real;
END_ENTITY;
```

Defines a direction towards the coordinates given by X, Y, Z.

Example:

```
OR3 (#or11:-3.5,0.,-0.5);
```

B.3.27 ORIENT (orient_name:direction_definition);

```
ENTITY orient;
   direction            : SELECT (OR1,OR3);
END_ENTITY;
```

Defines the orientation of 1D elements where necessary. orient_name may be referenced by BEAM2, SPRING, DAMPER and MASS.

The direction_definition can be OR1 (indicating that a node_name is given to specify the direction) or OR3 (indicating that 3 coordinates of an auxiliary node are given).

Example:

```
ORIENT (#Or88:#Or22);
       defines Orientation #Or88 in the direction towards the node given
       in the OR1 keyword #Or22.
```

B.3.28 MASS (name:ELEM_NUMBER,prop,p1,x-orient,y-orient);

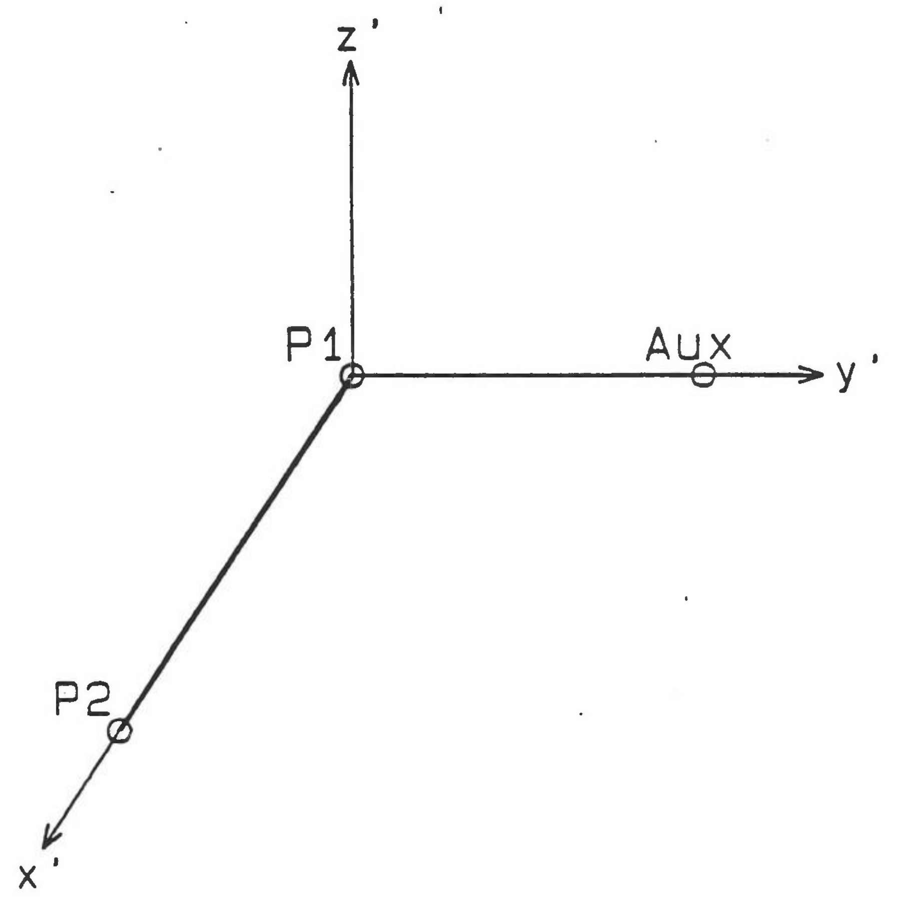
```
ENTITY mass;
   ELEM_NUMBER            : UNIQUE integer;
   prop                   : property;
   p1                     : node;
   x_orient               : OPTIONAL orient;
   y_orient               : OPTIONAL orient;
WHERE
   (* the directions defined by X_ORIENT and Y_ORIENT *)
   (* are at 90 degrees to each other.*)
END_ENTITY;
```

Defines a lumped mass at node p1; the values for mass and moments of inertia are given in a property statement. ELEM_NUMBER is the unique element number used by the FEM programs.

For translatory masses or if moments of inertia are given in global coordinates the orientations can be omitted, otherwise orientations must be given to define the directions of the local x and y axes.

Example:

```
MASS (#Ee:17,#P4,#N19,,);
       defines a lumped mass in node #N19 whose coefficients are given in
       #P4 in global coordinates.
```



Local coordinate system for all 1D elements

Element is defined by nodes P1 and P2
The local x'-axis lies along the element
Auxiliary point defines local y'-axis

Figure 6. Local Coordinate system for 1D elements.

B.3.29 DAMPER (name:ELEM_NUMBER,prop,p1,p2,orientation);

```
ENTITY damper;
   ELEM_NUMBER          : UNIQUE integer;
   prop                 : property;
   p1,p2                : node;
   orientation          : OPTIONAL orient;
END_ENTITY;
```

Defines a damper between nodes p1 and p2; its damping coefficients are given in the referenced property statement.

ELEM_NUMBER is the unique element number used by the FEM programs.

If orientation is omitted the damper is defined in global coordinates, otherwise the element x-axis goes from p1 to p2 and the element y-axis is defined by the referenced ORIENT statement.

The local element coordinate system is shown in figure 6.

Example:

```
DAMPER (#Ed:7,#P10,#N15,#N16,);
       defines damper #Ed as element number 7 with damping coefficients
       #P10 given in global coordinates.
```

B.3.30 SPRING (name:ELEM_NUMBER,prop,p1,p2,orientation);

```
ENTITY spring;
   ELEM_NUMBER          : UNIQUE integer;
   prop                 : property;
   p1,p2                : node;
   orientation          : OPTIONAL orient;
END_ENTITY;
```

Defines a spring between nodes p1 and p2; its stiffnesses are given in the referenced property statement.

elem_number is the unique element number used by the FEM programs.

If orientation is omitted the spring is defined in global coordinates, otherwise the element x-axis goes from p1 to p2 and the element y-axis is defined by the referenced ORIENT statement.

The local element coordinate system is shown in figure 6.

Example:

SPRING (#Ec:23,#P9,#N72,#N73,#Or5);
defines a spring element between nodes #N72 and #N73. The element x-axis goes from #N72 to #N73, the element y-axis from #N72 to the point given in #Or5. The stiffness coefficients are given in #P9.

B.3.31 BAR (name:ELEM_NUMBER,mat,prop,p1,p2);

```
ENTITY bar;
   ELEM_NUMBER          : UNIQUE integer;
   mat                  : material;
   prop                 : property;
   p1,p2                : node;
END_ENTITY;
```

Defines a bar element having stiffness only in longitudinal direction, ie. it has no rotational degrees of freedom.

ELEM_NUMBER is the element number used in the FEM programs, this number must be unique for all elements.

p1 and p2 are the names of the nodes at the ends of the bar;

mat and prop are the names of a material and of a property defined in separate statements.

Example:

BAR (#Ea: 21,#M1,#P2,#N341,#N12);
defines Element 21 as a bar (called #Ea) between nodes #N341 and #N12 with material (#M1) and properties (#P2) being defined elsewhere.

B.3.32 BEAM2 (name:ELEM_NUMBER,mat,prop,p1,p2,orientation);

```
ENTITY beam2;
   ELEM_NUMBER            : UNIQUE integer;
   mat                    : material;
   prop                   : property;
   p1,p2                  : node;
   orientation            : OPTIONAL orient;
END_ENTITY;
```

Defines a beam element between nodes p1 and p2 with resistance against tension/compression, bending, shear and torsion.

ELEM_NUMBER is the unique element number used by the FEM programs.

mat and prop are the names of a material and of a property defined in separate statements.

orientation points to an ORIENT statement which defines the direction of the element y-axis. (The element x-axis goes from p1 to p2.)

The local element coordinate system is shown in figure 6.

Example:

BEAM2 (#Eb:22,#M1,#P20,#N23,#N123,#Or88);
defines the 22nd element as a beam between nodes #N23 and #N123. The element y-axis valid for properties (defined in #P20) goes from #N23 to the point given in #Or88.

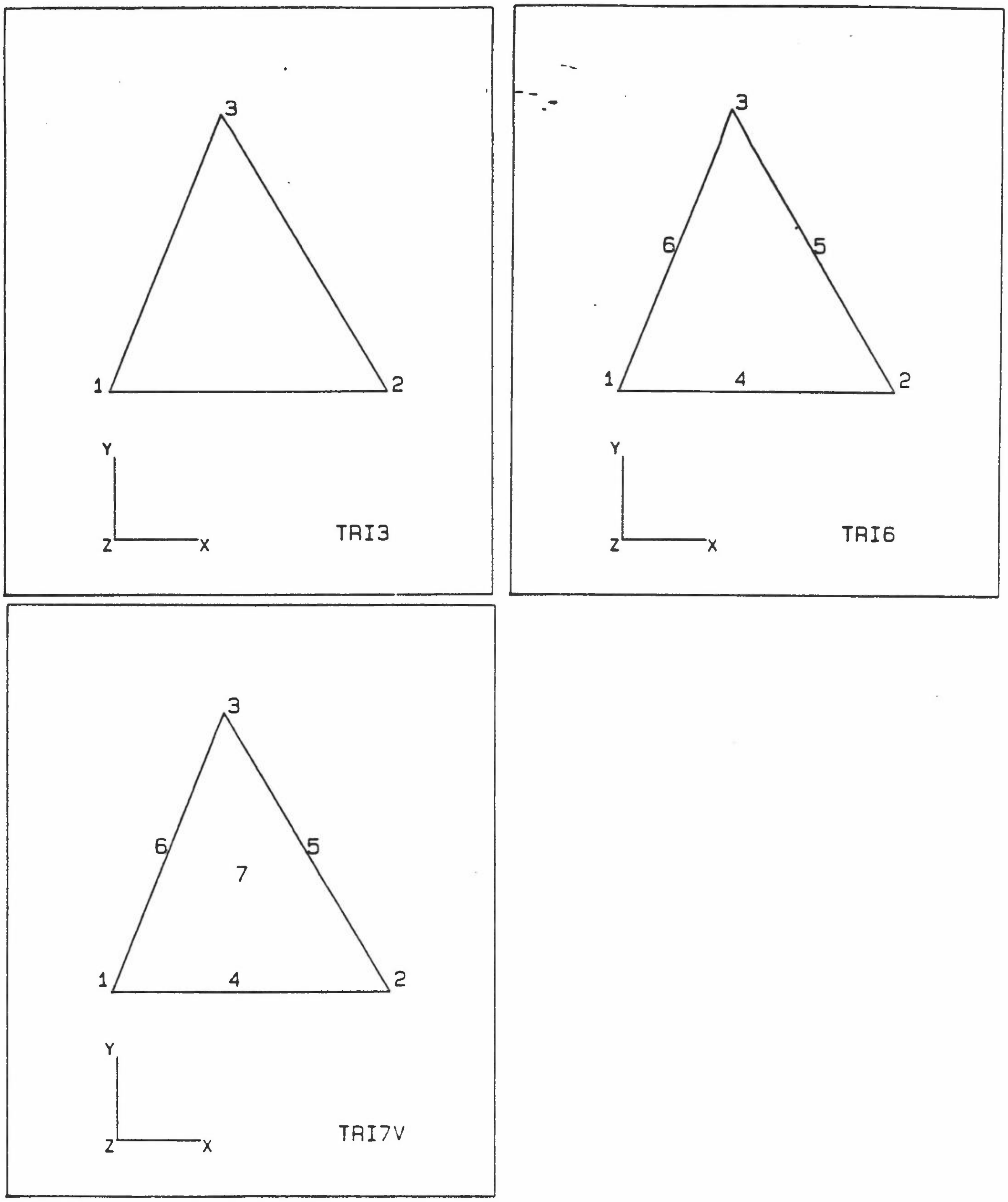


Figure 7. Node Ordering for Triangular 2D elements

B.3.33 TRI3 (name:ELEM_NUMBER,mat,prop,p1,p2,p3);

```
ENTITY tri3;
   ELEM_NUMBER          : UNIQUE integer;
   mat                  : material;
   prop                 : property;
   p1,p2,p3             : node;
END_ENTITY;
```

Defines a triangular 2D element with three nodes. Material and properties are given in separate statements.

ELEM_NUMBER is the unique element number used by the FEM programs.

Node ordering is shown in figure 7.

Example:

```
TRI3 (#Ef:24,#M2,#P11,#N1,#N2,#N3);
       defines triangle 24 made from material #M2 between nodes #N1, #N2
       and #N3. Thickness and purpose of the element are given in #P11.
```

B.3.34 TRI6 (name:ELEM_NUMBER,mat,prop,p1,p2,p3,p4,p5,p6);

```
ENTITY tri6;
   ELEM_NUMBER            : UNIQUE integer;
   mat                    : material;
   prop                   : property;
   p1,p2,p3,p4,p5,p6      : node;
END_ENTITY;
```

Defines a triangular 2D element with six nodes.

Material and properties are given in separate statements.

ELEM_NUMBER is the unique element number used by the FEM programs.

Node ordering is shown in figure 7.

Example:

```
TRI6 (#Eg:28,#M2,#P4,#N1,#N2,#N3,#N4,#N5,#N6);
       defines element 28 with name #Eg between the listed nodes.
       Thickness is given in #P4.
```

B.3.35 TRI7V (name:ELEM_NUMBER,mat,prop,p1,p2,p3,p4,p5,p6,p7);

```
ENTITY tri7v;
   ELEM_NUMBER          : UNIQUE integer;
   mat                  : material;
   prop                 : property;
   (* variable number of nodes *)
   p1,p2,p3             : node;
   p4,p5,p6,p7          : OPTIONAL node;
END_ENTITY;
```

Defines a triangular 2D element with up to 7 nodes. The 7th node is in the middle of the element. Nonexistent nodes can be omitted. The corner nodes (the first three) must always be supplied.

ELEM_NUMBER is the unique element number used by the FEM programs.

Material and properties are given in separate statements.

Node ordering is shown in figure 7.

Example:

```
TRI7V (#Eh:29,#M2,#P4,#N1,#N2,#N3,#N4,,#N6,);
```

defines element 29 with name #Eh between the listed nodes. The 5th and 7th node are omitted. Thickness is given in #P4.

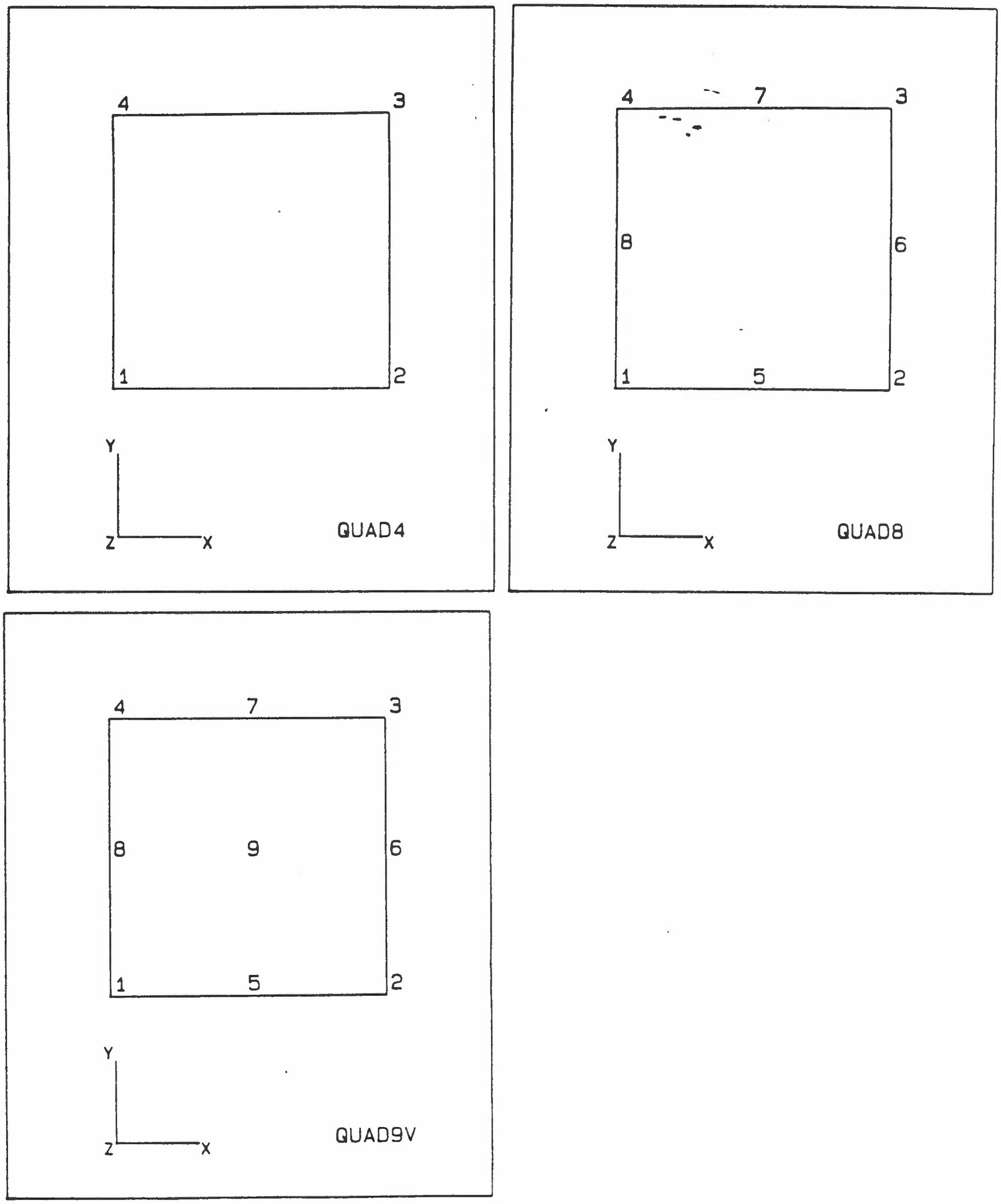


Figure 8. Node Ordering for Quadrilateral 2D elements.

B.3.36 QUAD4 (name:ELEM_NUMBER,mat,prop,p1,p2,p3,p4);

```
ENTITY quad4;
   ELEM_NUMBER          : UNIQUE integer;
   mat                  : material;
   prop                 : property;
   p1,p2,p3,p4          : node;
END_ENTITY;
```

Defines a quadrilateral 2D element with 4 nodes

ELEM_NUMBER is the unique element number used by the FEM programs.

mat and prop are the names of a material and of a property defined in separate statements.

Node ordering is shown in figure 8.

Example:

```
QUAD4 (#Ei:30,#M2,#P4,#N1,#N2,#N3,#N4);
       defines element 30 with name #Ei between the listed nodes.
       Thickness is given in #P4.
```

B.3.37 QUAD8 (name:ELEM_NUMBER,mat,prop,p1,p2,p3,p4,p5,p6,p7,p8);

```
ENTITY quad8;
   ELEM_NUMBER            : UNIQUE integer;
   mat                    : material;
   prop                   : property;
   p1,p2,p3,p4,p5,p6      : node;
   p7,p8                  : node;
END_ENTITY;
```

Defines a quadrilateral 2D element with 8 nodes.

ELEM_NUMBER is the unique element number used by the FEM programs.

mat and prop are the names of a material and of a property defined in separate statements.

Node ordering is shown in figure 8.

Example:

```
QUAD8 (#Ej:31,#M2,#P4,#N1,#N2,#N3,#N4,#N5,#N6,#N7,#N8);
       defines element 31 with name #Ej between the listed nodes.
       Thickness is given in #P4.
```

B.3.38 QUAD9V (name:ELEM_NUMBER,mat,prop,p1,p2,p3,p4,p5,p6,p7,p8,p9);

```
ENTITY quad9v;
   ELEM_NUMBER          : UNIQUE integer;
   mat                  : material;
   prop                 : property;
   (* variable number of nodes *)
   p1,p2,p3,p4          : node;
   p5,p6,p7,p8          : OPTIONAL node;
   p9                   : OPTIONAL node;
END_ENTITY;
```

Defines a quadrilateral 2D element with up to 9 nodes. First the corner nodes are given, then (optionally) the edge nodes, then (optionally) the centre node.

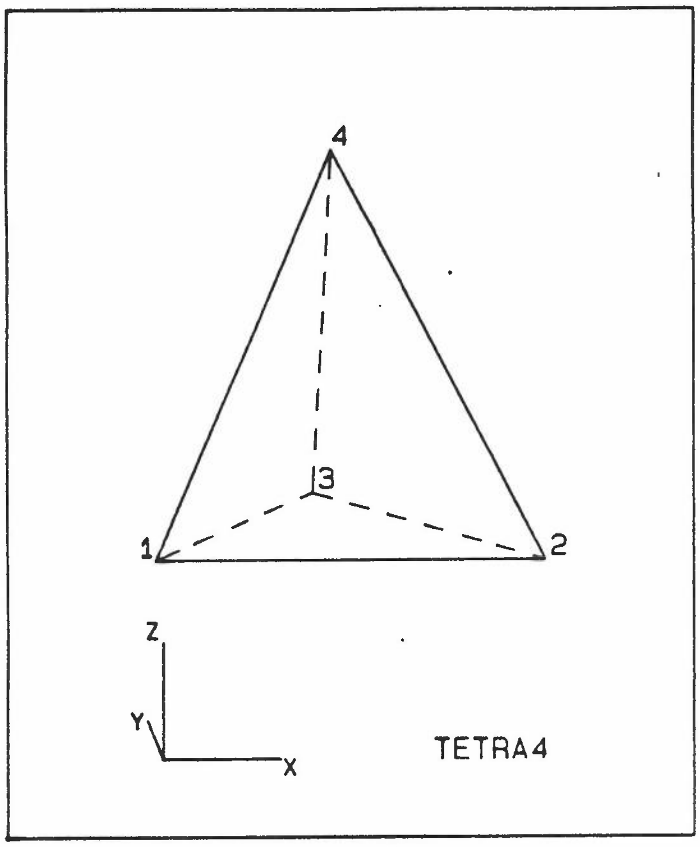
ELEM_NUMBER is the unique element number used by the FEM programs.

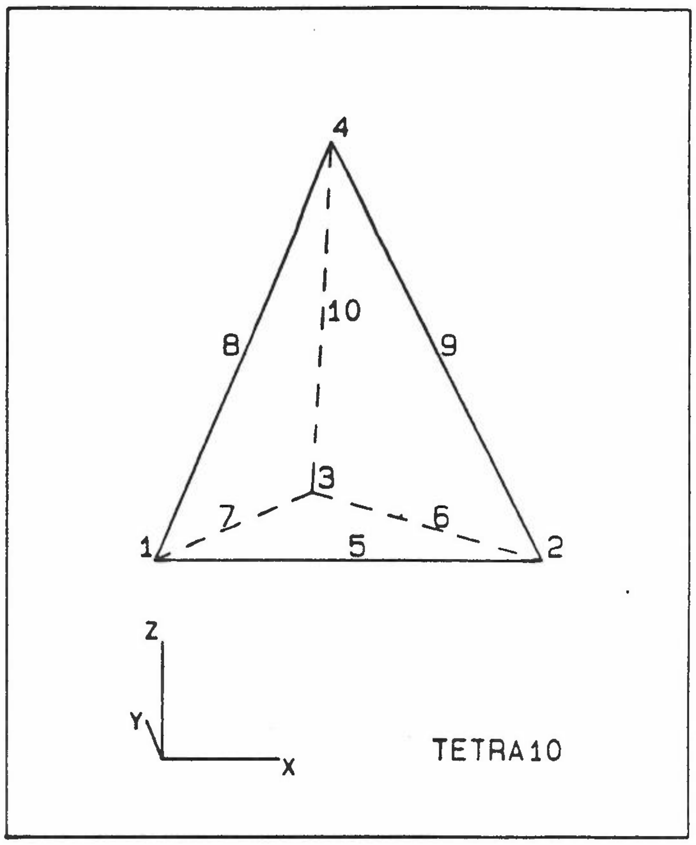
mat and prop are the names of a material and of a property defined in separate statements.

Node ordering is shown in figure 8.

Example:

QUAD9V (#Ek:32,#M2,#P4,#N1,#N2,#N3,#N4,,,,,#N9);
defines element 32 with name #Ek between the listed nodes, 4 corner nodes and one in the middle. Thickness is given in #P4.





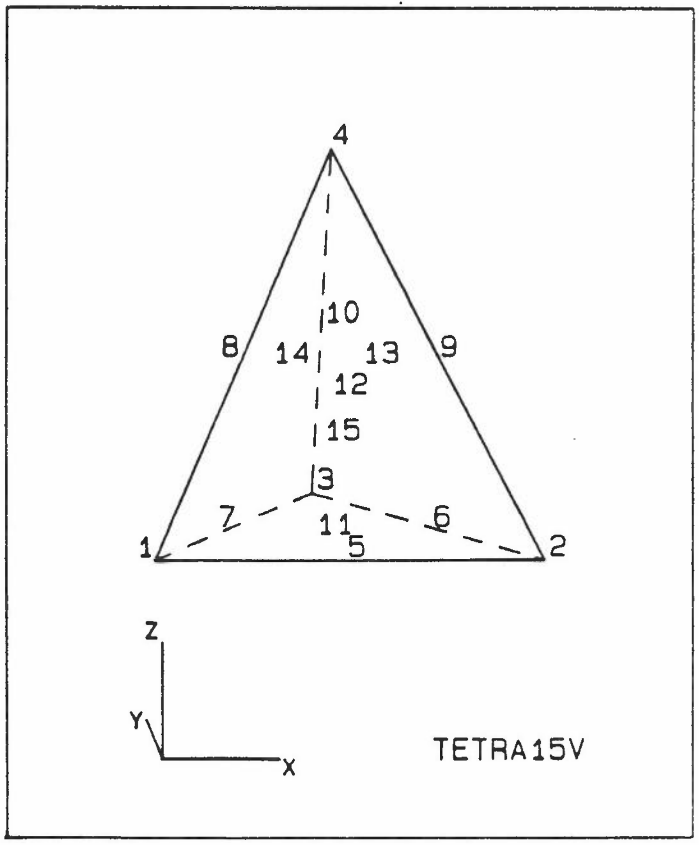


Figure 9. Node Ordering for Tetrahedron solid elements

B.3.39 TETRA4 (name:ELEM_NUMBER,mat,prop,p1,p2,p3,p4);

```
ENTITY tetra4;
   ELEM_NUMBER             : UNIQUE integer;
   mat                     : material;
   prop                    : OPTIONAL property;
   p1,p2,p3,p4             : node;
END_ENTITY;
```

Defines a tetrahedron volume element by its 4 corner nodes. The material is given in a separate statement, prop may be omitted (later the property statement may define an integration control parameter).

ELEM_NUMBER is the unique element number used by the FEM programs.

mat and prop are the names of a material and of a property defined in separate statements.

Node ordering is shown in figure 9.

Example:

```
TETRA4 (#E1:33,#M2,,#N1,#N2,#N3,#N4);
       defines element 33 with name #E1 between the listed nodes. The
       material is given in #M2.
```

B.3.40 TETRA10 (name:ELEM_NUMBER,mat,prop,p1,p2,p3,p4,p5,p6,p7,p8,p9,p10);

```
ENTITY tetra10;
   ELEM_NUMBER           : UNIQUE integer;
   mat                   : material;
   prop                  : OPTIONAL property;
   p1,p2,p3,p4,p5,p6     : node;
   p7,p8,p9,p10          : node;
END_ENTITY;
```

Defines a tetrahedron volume element with one additional node on each edge.

ELEM_NUMBER is the unique element number used by the FEM programs.

mat and prop are the names of a material and of a property defined in separate statements.

Node ordering is shown in figure 9.

Example:

```
TETRA10 (#Em:34,#M2,,#N101,#N102,#N103,#N104,#N105,#N106,#N107,
            #N108,#N109,#N110);
```

defines element 34 with name #Em between the listed nodes. The material is given in #M2.

B.3.41 TETRA15V (name:ELEM_NUMBER,mat,prop,p1,p2,p3,...,p15);

```
ENTITY tetra15v;
   ELEM_NUMBER           : UNIQUE integer;
   mat                   : material;
   prop                  : OPTIONAL property;
   (* variable number of nodes *)
   p1,p2,p3,p4           : node;
   p5,p6,p7,p8           : OPTIONAL node;
   p9,p10,p11            : OPTIONAL node;
   p12,p13,p14           : OPTIONAL node;
   p15                   : OPTIONAL node;
END_ENTITY;
```

Defines a tetrahedron volume element with up to 15 nodes. First the 4 corner nodes are given, then (optionally) 6 nodes on the edges, then (optionally) 4 nodes on the surfaces, then (optionally) 1 node in the middle of the body.

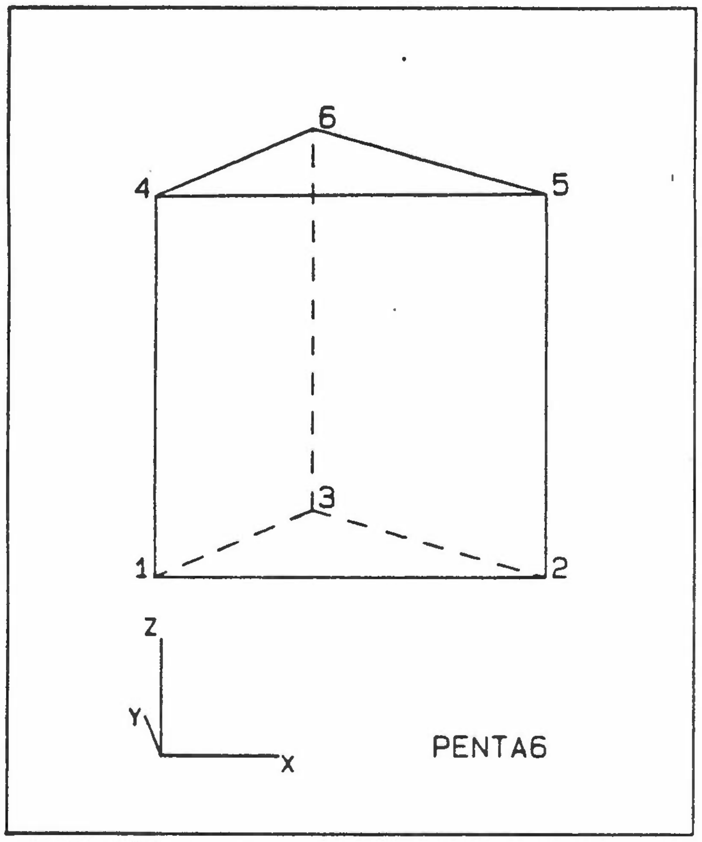
ELEM_NUMBER is the unique element number used by the FEM programs.

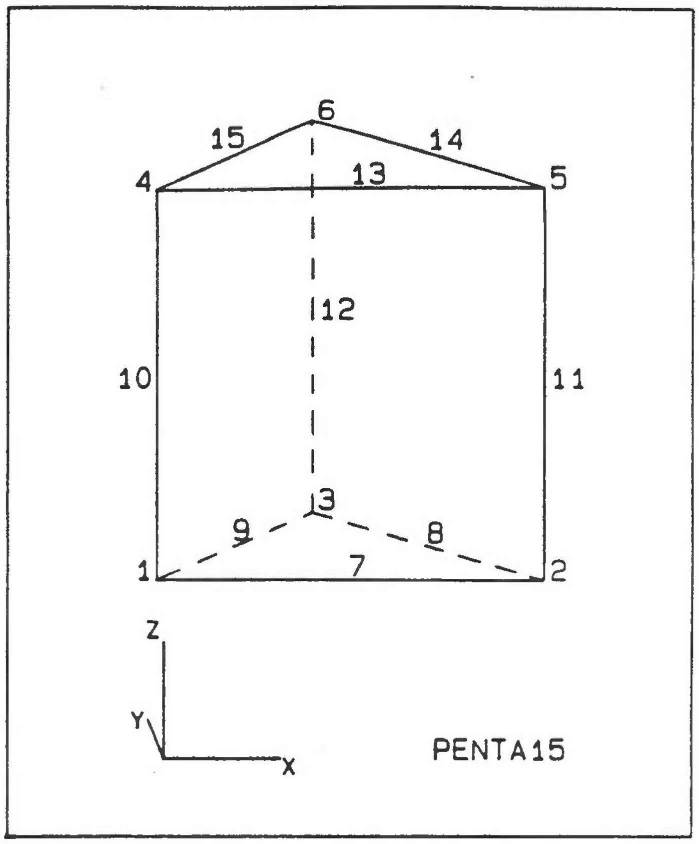
mat and prop are the names of a material and of a property defined in separate statements.

Node ordering is shown in figure 9.

Example:

```
TETRA15V (#En:35,#M2,,#N101,#N102,#N103,#N104,,,,,,,,,,,#N115);
       defines element 35 with name #En between the listed nodes, 4 at the
       corners and one in the middle. The material is given in #M2.
```





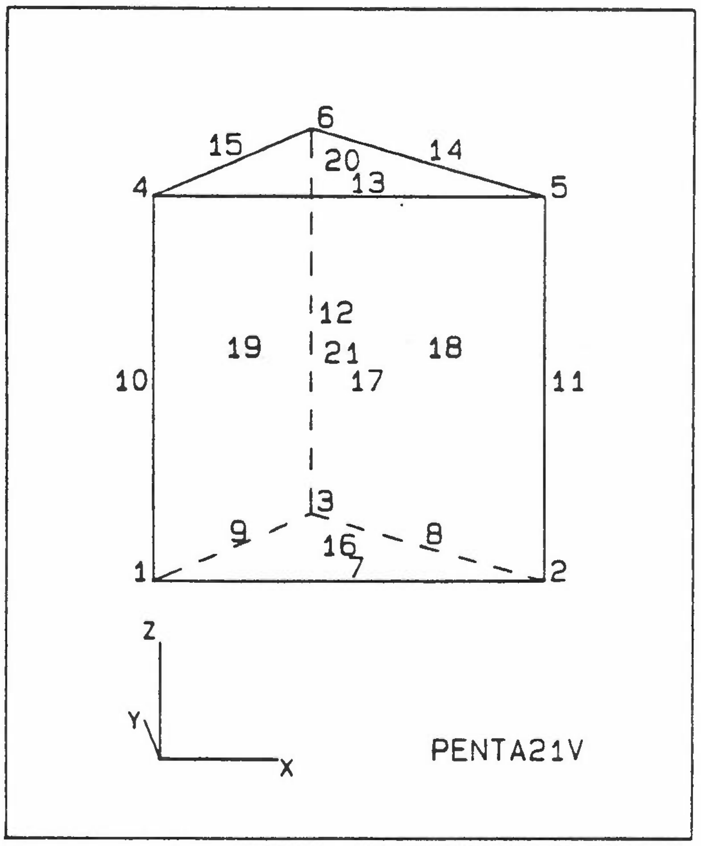


Figure 10. Node Ordering for Pentahedron solid elements.

B.3.42 PENTA6 (name:ELEM_NUMBER,mat,prop,p1,p2,p3,p4,p5,p6);

```
ENTITY penta6;
   ELEM_NUMBER            : UNIQUE integer;
   mat                    : material;
   prop                   : OPTIONAL property;
   p1,p2,p3,p4,p5,p6      : node;
END_ENTITY;
```

Defines a pentahedron volume element by its 6 corner nodes. The material is given in a separate statement, prop may be omitted (later the property statement may define an integration control parameter).

ELEM_NUMBER is the unique element number used by the FEM programs.

mat and prop are the names of a material and of a property defined in separate statements.

Node ordering is shown in figure 10.

Example:

```
PENTA6 (#Eo:36,#M2,,#N71,#N72,#N73,#N74,#N75,#N76);
       defines element 36 with name #Eo between the listed nodes.
```

B.3.43 PENTA15 (name:ELEM_NUMBER,mat,prop,p1,p2,p3,...,p15);

```
ENTITY penta15;
   ELEM_NUMBER          : UNIQUE integer;
   mat                  : material;
   prop                 : OPTIONAL property;
   p1,p2,p3,p4,p5,p6    : node;
   p7,p8,p9,p10,p11     : node;
   p12,p13,p14,p15      : node;
END_ENTITY;
```

Defines a pentahedron volume element with one additional node on each edge

ELEM_NUMBER is the unique element number used by the FEM programs.

mat and prop are the names of a material and of a property defined in separate statements.

Node ordering is shown in figure 10.

Example:

```
PENTA15 (#Ep:37,#M2,,#N71,#N72,#N73,#N74,#N75,#N76,#N77,#N78,#N79,#N80,
            #N81,#N82,#N83,#N84,#N85);
       defines element 37 with name #Ep between the listed nodes.
```

B.3.44 PENTA21V (name:ELEM_NUMBER,mat,prop,p1,p2,p3,...,p21);

```
ENTITY penta21v;
   ELEM_NUMBER            : UNIQUE integer;
   mat                    : material;
   prop                   : OPTIONAL property;
   (* variable number of nodes *)
   p1,p2,p3,p4            : node;
   p5,p6                  : node;
   p7,p8                  : OPTIONAL node;
   p9,p10,p11             : OPTIONAL node;
   p12,p13,p14            : OPTIONAL node;
   p15,p16,p17            : OPTIONAL node;
   p18,p19,p20            : OPTIONAL node;
   p21                    : OPTIONAL node;
END_ENTITY;
```

Defines a pentahedron volume element with up to 21 nodes. First the 6 corner nodes are given, then (optionally) 9 nodes on the edges, then (optionally) 5 nodes on the surfaces, then (optionally) 1 node in the middle of the body.

ELEM_NUMBER is the unique element number used by the FEM programs.

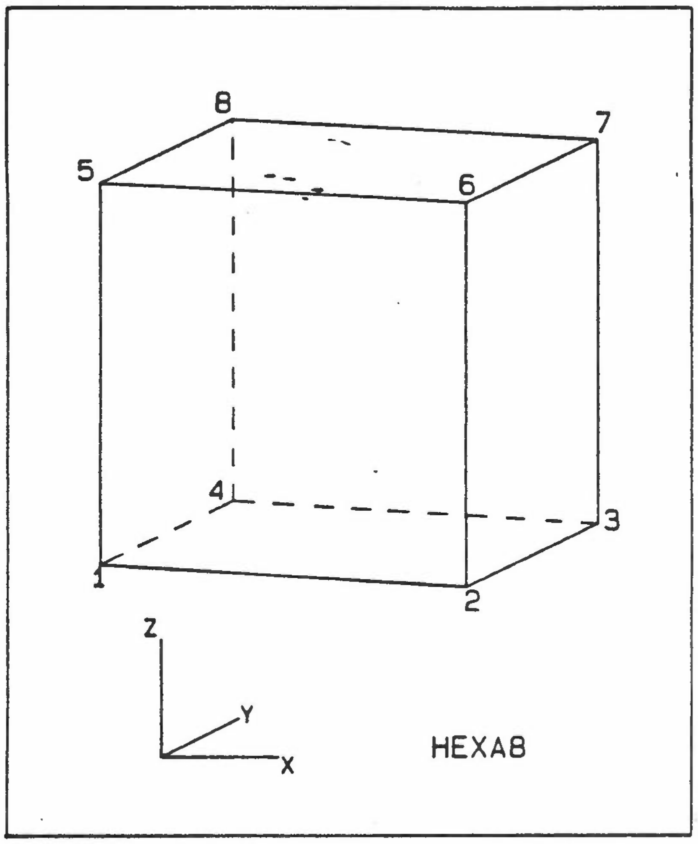
mat and prop are the names of a material and of a property defined in separate statements.

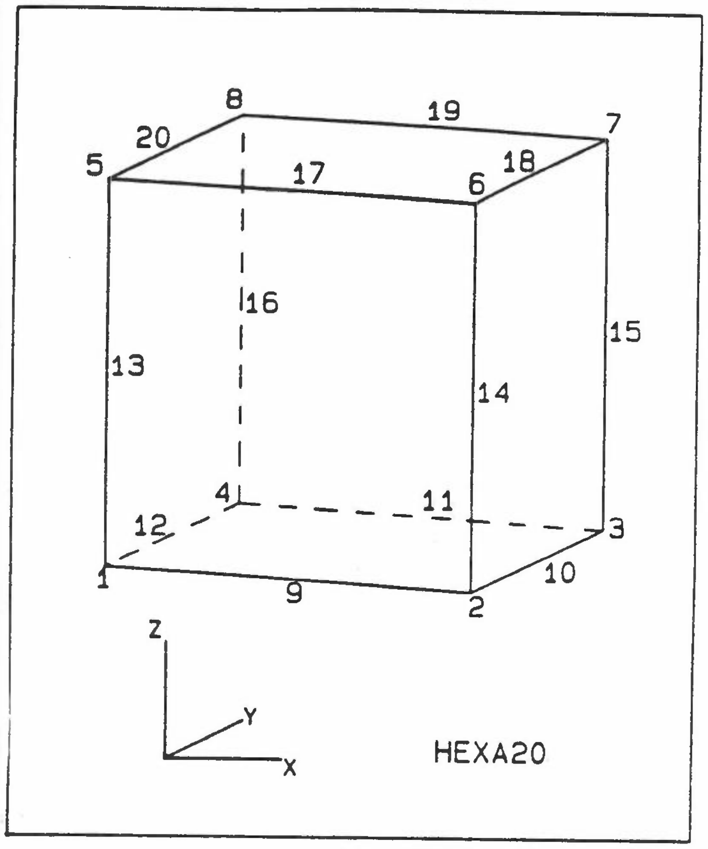
Node ordering is shown in figure 10.

Example:

```
PENTA21V (#Eq:38,#M2,,#N71,#N72,#N73,#N74,#N75,#N76,#N77,#N78,#N79,#N80,
          #N81,#N82,#N83,#N84,#N85,#N86,#N87,#N88,#N89,#N90,#N91);
```

defines element 38 with name #Eq between the listed nodes. All nodes are given: on corners, edges, surfaces and in the middle.





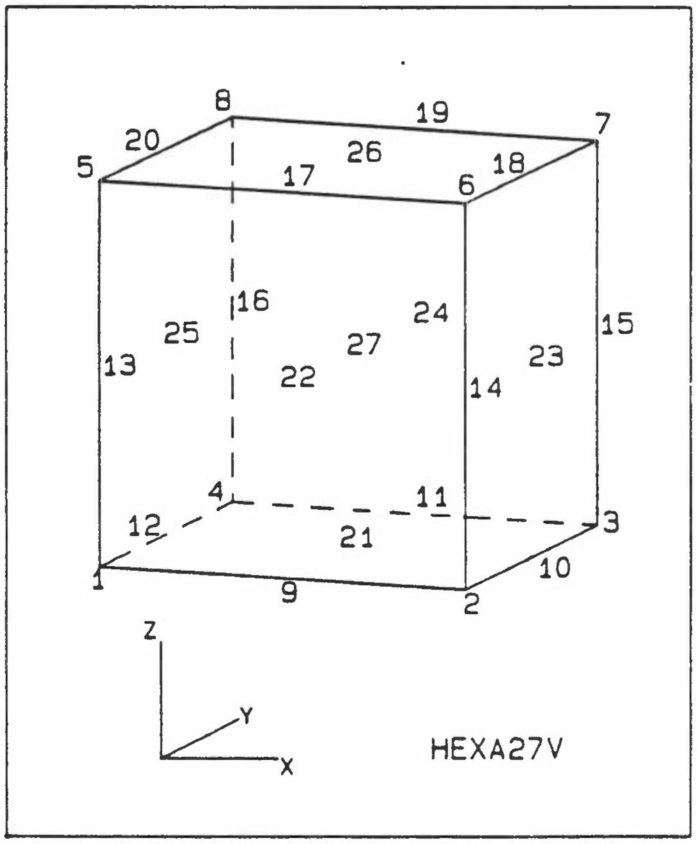


Figure 11. Node Ordering for Hexahedron solid elements.

B.3.45 HEXA8 (name:ELEM_NUMBER,mat,prop,p1,p2,p3,p4,p5,p6,p7,p8);

```
ENTITY hexa8;
   ELEM_NUMBER          : UNIQUE integer;
   mat                  : material;
   prop                 : OPTIONAL property;
   p1,p2,p3,p4          : node;
   p5,p6,p7,p8          : node;
END_ENTITY;
```

Defines a hexahedron volume element by its 8 corner nodes. The material is given in a separate statement, prop may be omitted (later the property statement may define an integration control parameter).

ELEM_NUMBER is the unique element number used by the FEM programs.

mat and prop are the names of a material and of a property defined in separate statements.

Node ordering is shown in figure 11.

Example:

```
HEXA8 (#Er:39,#M2,,#N81,#N82,#N83,#N84,#N85,#N86,#N87,#N88);
       defines element 39 with name #Er between the listed nodes.
```

B.3.46 HEXA20 (name:ELEM_NUMBER,mat,prop,p1,p2,p3,...,p20);

```
ENTITY hexa20;
   ELEM_NUMBER             : UNIQUE integer;
   mat                     : material;
   prop                    : OPTIONAL property;
   p1,p2,p3,p4,p5,p6       : node;
   p7,p8,p9,p10,p11        : node;
   p12,p13,p14,p15,p16     : node;
   p17,p18,p19,p20         : node;
END_ENTITY;
```

Defines a hexahedron volume element with one additional node on each edge

ELEM_NUMBER is the unique element number used by the FEM programs.

mat and prop are the names of a material and of a property defined in separate statements.

Node ordering is shown in figure 11.

Example:

```
HEXA20 (#Es:40,#M2,,#N71,#N72,#N73,#N74,#N75,#N76,#N77,#N78,#N79,#N80,#N81,
          #N82,#N83,#N84,#N85,#N86,#N87,#N88,#N89,#N90);
       defines element 40 with name #Es between the listed nodes.
```

B.3.47 HEXA27V (name:ELEM_NUMBER,mat,prop,p1,p2,p3,...,p27);

```
ENTITY hexa27v;
   ELEM_NUMBER          : UNIQUE integer;
   mat                  : material;
   prop                 : OPTIONAL property;
   (* variable number of nodes *)
   p1,p2,p3,p4          : node;
   p5,p6,p7,p8          : OPTIONAL node;
   p9,p10,p11           : OPTIONAL node;
   p12,p13,p14          : OPTIONAL node;
   p15,p16,p17          : OPTIONAL node;
   p18,p19,p20          : OPTIONAL node;
   p21,p22,p23          : OPTIONAL node;
   p24,p25,p26          : OPTIONAL node;
   p27                  : OPTIONAL node;
END_ENTITY;
```

Defines a hexahedron volume element with up to 27 nodes. First the 8 corner nodes are given, then (optionally) 12 nodes on the edges, then (optionally) 6 nodes on the surfaces, then (optionally) 1 node in the middle of the body.

ELEM_NUMBER is the unique element number used by the FEM programs.

mat and prop are the names of a material and of a property defined in separate statements.

Node ordering is shown in figure 11.

Example:

```
HEXA27V (#Et:41,#M2,,#N81,#N72,#N63,#N54,#N45,#N36,#N27,#N18,#N9,#N19,#N29,
              #N39,#N49,#N59,#N69,#N79,#N89,#N99,#N109,#N119, ,,,,,,#N200);
```

defines element 41 with name #Et between the listed nodes. All corner and edge nodes are given, but no surface node; the middle node is given, too.

B.3.48 CLOSEMODEL (model_name);

```
PROPERTY closemodel;
OF
   model_name            : openmodel;
END_PROPERTY;
```

Closes the description of the finite element model.

Example:

```
CLOSEMODEL (#testmodel);
```

B.4 Keywords for the FE Analysis

B.4.1 OPENANA (analysis_name:model_name,'TEXT_DESCRIPTION');

```
ENTITY openana;
   model_name           : openmodel;
   TEXT                 : OPTIONAL string;
END_ENTITY;
```

Opens an analysis block belonging to a model. model_name was defined in OPENMODEL, analysis_name may be referenced by the result sets. The text is supplied by the user and may be omitted.

Example:

```
OPENANA (#ANALY1: #testmodel, '')
```

B.4.2 COLLECT (name:'OBJECT_DESCRIPTION',(list_of_elements));

```
ENTITY collect;
   OBJECT_DESCRIPTION    : OPTIONAL string;
   list_of_elements      : SET OF element;
END_ENTITY;

TYPE
   element = SELECT ( bar, beam2, spring, damper, mass, tri3,
                      tri6, tri7v, quad4, quad8, quad9v, tetra4,
                      tetra10, tetra15v, penta6, penta15,
                      penta21v, hexa8, hexa20, hexa21v );
END_TYPE;
```

Forms an object from the elements contained in (list_of_elements). This object can be manipulated as one unit for the purposes of copying, assembling, condensation and analysis.

'OBJECT_DESCRIPTION' is a string constant containing a textual description of the object. This may be omitted.

Example:

```
COLLECT (#OBJ1: 'object from elements #Ea...#Ej and #E5...
         #E10',(#Ea,#Eb,#Ec,#Ed,#Ee,#Ef,#Eg,#Eh,#Ei,#Ej,
         #E5,#E6,#E7,#E8,#E9, #E10) );
```

B.4.3 CDOFLIST (name:(node1,node2....,nodeNN), (DOF1,DOF2....DOFNN));

```
ENTITY cdoflist;
   node_list            : LIST [1 TO *] OF gnode;
   DOFS_LIST            : LIST [1 TO *] OF string;
WHERE
   SIZEOF (NODE_LIST) = SIZEOF (DOFS_LIST);
   (* Also many restrictions on the contents of the strings *)
END_ENTITY;

TYPE
   gnode = SELECT (node, trn);
END_TYPE;
```

Defines a list of degrees-of-freedom for the static condensation. NN is the number of members of both the list of nodes and the list of degrees of freedom. Only nodes from which degrees-of-freedom are to be removed must be mentioned, so NN may be less than the number of nodes in the structure.

node is a node name or tree node name and DOFS is a string containing up to 6 numeric characters denoting the degrees-of-freedom to be removed. Degrees-of-freedom which are already fixed should not appear in the CDOFLIST statement. Note that degrees-of-freedom with prescribed deformations or to which loads are attached must not be removed.

Examples:

'123456'	remove all 6 degrees of freedom
'456'	remove only rotational degrees of freedom
'3'	remove only 3rd degree of freedom (usually z)

Example:

```
CDOFLIST (#list: (#N41,#N42,#N43,#N44,#N50,#N51,#N52,#N61),
         ('123456','123456','123456','123456',
         '456','456','456','456'));
```

removes the rotations from nodes #N50,#N51,#N52 and #N61 and removes nodes #N41, #N42, #N43, #N44 completely.

B.4.4 SCOND (new_object:old_object,cdoflist);

```
ENTITY scond;
   old_object           : object;
   cdoflist             : cdoflist;
END_ENTITY;
TYPE
   object = SELECT (collect, assem, scond, copy);
END_TYPE;
```

Performs a static condensation (Guyan's reduction) on the old_object yielding the new_object.

The list of degrees-of-freedom which will be removed by the static condensation must be previously defined in a CDOFLIST.

Example:

```
SCOND (#OB2: #OB1, #list);
       performs a static condensation on object #OB1. The list of freedoms
       to be removed is #list. A new object #OB2 is created.
```

B.4.5 DCOND (new_object:eigname,(list_of_modes));

```
ENTITY dcond;
   eigname              : SELECT (anreigv,anceigv);
   list_of_modes        : SET OF integer;
END_ENTITY;
```

Performs a dynamic condensation (i.e. removal of eigenvectors on the object mentioned in the ANREIGV/ANCEIGV statement). eigname identifies the ana-id of the real or complex eigenvalue problem.

list_of_modes is a sequence of integer numbers telling which eigenvectors should be used in modal superposition.

Example:

DCOND (#OB5: #ana2,(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17);
performs a dynamic condensation using the first 17 modes of analysis #ana2 (based on the object mentioned there) and produces object #OB5 (which is not a physical object but only a set of eigenvectors).

B.4.6 COUPLELIST (couple_list_name:(nodal,...nodaN), (nodebl,...nodebN));

```
ENTITY couplelist;
   nodes_a              : LIST [1 TO *] OF gnode;
   nodes_b              : LIST [1 TO *] OF gnode;
WHERE
   SIZEOF (nodes_a) = SIZEOF (nodes_b);
END_ENTITY;

TYPE
   gnode = SELECT (node, trn);
END_TYPE;
```

Defines a list of N coupling point pairs for the assembling step. Normally this is not necessary. It must be used only for those coupling nodes whose coordinates differ more than the specified tolerance in the ASSEM statement.

Example:

```
COUPLELIST (#listA: (#N1,#N2,#N3),(#N4,#N5,#N6));
       defines a list to be referenced by an ASSEM statement which tells,
       that nodes #N1 and #N4, #N2 and #N5, #N3 and #N6 are to be coupled.
```

B.4.7 NOCOUPLE (nocouple_name:(node1,node2..nodeN), (DOF1,DOF2..DOFN));

```
ENTITY nocouple;
   nodes                : LIST [1 TO *] OF gnode;
   DOFS                 : LIST [1 TO *] OF string;
WHERE
   SIZEOF (nodes) = SIZEOF (DOFS);
   (* Also restrictions on contents of string *)
END_ENTITY;

TYPE
   gnode = SELECT (node, trn);
END_TYPE;
```

Flags nodes or treenodes of which the specified degree-of-freedom must not be coupled in the assemble step. DOF is a string containing those degrees of freedom which should not be coupled.

Example:

```
NOCOUPLE (#listB: (#N7,#N8,#N9,#N10), ('123456','123456','456', '456'));
```

defines a list to be referenced by an ASSEM statement which tells, that nodes #N7 and #N8 are never to be coupled, and that of nodes #N9 and #N10 the rotations are not to be coupled.

B.4.8 COPY (new_object:old_object,cosys);

```
ENTITY copy;
   old_object           : object;
   cosys                : coord;
END_ENTITY;

TYPE
   object = SELECT (collect, assem, scond, copy);
END_TYPE;
```

Creates a copy of an existing object which can be used in the substructure assembly. The parameter cosys describes the transformation to be performed (shift, rotation, mirroring). Note that this is not a geometrical modelling but only a matrix transformation which is used for multiple insertion of substructures into a main structure.

Example:

```
COPY (#OB3: #OB2,#co5);
       creates the new object #OB3 from the old object #OB2, the geometric
       transformation is defined by the coordinate system #co5.
```

B.4.9 ASSEM (new_obj:TOL,couple_list_name,nocouple_name, (list_of_obj_names));

```
ENTITY assem;
   TOL                   : real;
   couple_list_name      : OPTIONAL couplelist;
   nocouple_name         : OPTIONAL nocouple;
   object_list           : SET OF object;
END_ENTITY;

TYPE
   object = SELECT (collect, copy, assem, scond);
END_TYPE;
```

performs a substructure assembly, the substructures given in the list of obj_names are joined together forming the new (main) structure new_obj.

Nodes of the old objects having (within the given tolerance, TOL) the same coordinates are coupled with all degrees-of-freedom. If this is not desired, the respective nodes must appear in a nocouple entity. nocouple_name is omitted if no special uncoupling is required.

The couple_list_name refers to a couplelist entity which contains a list of nodes which become identical between the substructures even if they are further apart than the given tolerance (parameter TOL). The parameter couple_list_name is omitted if no coupling outside tolerance is required.

Examples:

```
ASSEM (#OB6: 1.E-4,,,(#OB3,#OB4,#OB5);
```

creates the new object #OB6 from the existing objects #OB3, #OB4, and #OB5. Common nodes will be recognized if they are not further than 1.E-4 apart.

```
ASSEM (#OB6: 1.E-4, COUPLELIST((#N19),(#N222)), NOCOUPLE((#N2,#N3,#N4),
      ('123456','123456','123456')), (#OB3,#OB4,#OB5);
```

performs the same assembly, but nodes #N19 and #N222 will be joined in one node even if the given tolerance would be smaller than their distance. Nodes #N2, #N3, #N4 will never be coupled, even if other nodes are lying closer to them than the tolerance.

B.4.10 TRN (name:object,node);

```
ENTITY trn;
   object               : SELECT ( copy, assem );
   node                 : gnode;
END_ENTITY;

TYPE
   gnode = SELECT ( node, trn);
END_TYPE;
```

This keyword allows the possibility of referencing a node which has been created using the copy and assembly commands and which, therefore, has been given no name.

object is the name of an object formed from a copy or assembly.

node is either a node name or another trn, which allows for nesting of objects.

Examples:

```
TRN (#NN15: #OB2,#N15);
       defines a node #NN15 in the copied object #OB2. It was created from
       the node #N15 in the original object.

TRN (#NO123: #OB3, TRN(#OB2, TRN(#OB1,#N2) ) );
       shows how to identify nodes in objects copied from copied objects.
```

B.4.11 TRE (name:object,element);

```
ENTITY tre;
   object              : SELECT ( copy, assem );
   element             : gelement;
END_ENTITY;

TYPE
   gelement = SELECT ( element, tre );
   element = SELECT ( bar, beam2, spring, damper, mass, tri3,
                      tri6, tri7v, quad4, quad8, quad9v, tetra4,
                      tetra10, tetra15v, penta6, penta15,
                      penta21v, hexa8, hexa20, hexa21v );
END_TYPE;
```

This keyword allows the possibility of referencing an element which has been created using the copy and assembly commands and which, therefore, has been given no name.

object is the name of an object formed from a copy or assembly.

element is either an element name or another tre, which allows for nesting of objects.

Examples:

TRE (#EL15: #OB2,#E15);
        defines an element #EL15 in the copied object #OB2. It was created from the element #E15 in the original object.

TRE (#ELE123: #OB3, TRE(#OB2, TRE(#OB1,#E2)));
        shows how to identify elements in objects copied from copied objects.

B.4.12 ALLNODES(name: (list_of_objects));

```
ENTITY allnodes;
   list_of_objects        : SET OF object;
END_ENTITY;

TYPE
   object = SELECT (collect, assem, scond, copy);
END_TYPE;
```

Allows reference to be made to all the nodes contained in the objects listed.

Example:

```
ALLNODES (#list3:(#OB1));
       defines a name (#list3) for all nodes of object #OB1.
```

B.4.13 ALLELEMS(name: (list_of_objects));

```
ENTITY allelems;
   list_of_objects        : SET OF object;
END_ENTITY;

TYPE
   object = SELECT (collect, assem, scond, copy);
END_TYPE;
```

Allows reference to be made to all the elements contained in the objects listed.

Example:

```
ALLELEMS (#list4:(#OB1,#OB2,#OB3));
       defines a name (#list4) for all elements of objects #OB1, #OB2, and
       #OB3.
```

B.4.14 NODELIST(name:(list_of_nodes));

```
ENTITY nodelist;
   list_of_nodes        : SET OF gnode;
END_ENTITY;

TYPE
   gnode = SELECT (node, trn);
END_TYPE;
```

Allows reference to be made to a list of nodes.
The list_of_nodes may contain both nodes and treenodes. For sequencing of matrices this statement may also occur in the result section of the neutral file.

Example:

```
NODELIST (#list1: (#N5,#N6,#N7,#N8) );
       gives the listed nodes the name #list1 which can be referenced.
```

B.4.15 ELEMLIST(name:(list_of_elements));

```
ENTITY elemlist;
   list_of_elements     : SET OF gelement;
END_ENTITY;

TYPE
   gelement = SELECT ( element, tre );
   element = SELECT ( bar, beam2, spring, damper, mass, tri3,
                      tri6, tri7v, quad4, quad8, quad9v, tetra4,
                      tetra10, tetra15v, penta6, penta15,
                      penta21v, hexa8, hexa20, hexa21v );
END_TYPE;
```

Allows reference to be made to a list of elements.
List_of_elements may contain both elements and tree elements. For sequencing of matrices this statement may also occur in the result section of the neutral file.

Example:

```
ELEMLIST (#list2: (#E5,#E6,#E7,#E8) );
       gives the listed elements the name #list2 which can be referenced.
```

B.4.16 NODAL (name:TYPE_CODE,nodal_reference,cosys);

```
ENTITY nodal;
   TYPE_CODE            : INTEGER;
   nodal_reference      : SELECT (gnode,nodelist,allnodes);
   cosys                : OPTIONAL coord;
WHERE;
   TYPE_CODE >= 1 AND TYPE_CODE <= 4;
END_ENTITY;

TYPE
   gnode = SELECT (node, trn);
END_TYPE;
```

Selects nodes for FE analysis output requests for stresses and strains.

cosys is the optional coordinate system in which results should be output. The default is the global cartesian system.

type_code is an integer number to enable the choice between different stress options;

1	6 stress components
2	principal stresses (3 stress values, 9 direction cosines)
3	von Mises equivalent stresses, one value per node
4	Tresca equivalent stresses, one value per node

For strains the same options apply.

Example:

```
NODAL (#ndl3:3,#alll, );
```

allows a request for von Mises equivalent stresses for the nodes referenced by #alll, in the global cartesian coordinate system.

B.4.17 ELEMENTAL (name:TYPE_CODE,element_reference,cosys);

```
ENTITY elemental;
   TYPE_CODE               :INTEGER;
   element_reference       : SELECT(gelement,elemlist,allelems);
   cosys                   : OPTIONAL coord;
WHERE;
   TYPE_CODE = 1;
END_ENTITY;

TYPE
   gelement = SELECT ( element, tre );
   element  = SELECT ( bar, beam2, spring, damper, mass, tri3,
                        tri6, tri7v, quad4, quad8, quad9v, tetra4,
                        tetra10, tetra15v, penta6, penta15,
                        penta21v, hexa8, hexa20, hexa21v );
END_TYPE;
```

Selects elements for FE analysis output requests for stresses or strains.

cosys is the optional coordinate system in which results should be output. The default is the global cartesian system.

type_code is an integer number to enable the choice between different stress or strain options (see NODAL, B.4.11), however, only the option TYPE_CODE = 1 is currently used.

Example:

ELEMENTAL (#emntl:1,#box,#coor3)
allows a request for elemental stresses of the elements referenced by the entity #box, in the coordinate system #coor3.

B.4.18 PRESET1 (nodal_reference,IDIR,DEF);

```
PROPERTY preset1;
   IDIR                 : integer;
   DEF                  : real;
OF
   nodal_reference      :SELECT ( gnode, nodelist, allnodes);
WHERE
    IDIR >= 1 AND IDIR <= 6;
END_PROPERTY;

TYPE
   gnode = SELECT (node, trn);
END_TYPE;
```

Assigns a prescribed displacement DEF in direction IDIR to the nodes given in nodal_reference. This parameter may refer to a NODE, TRN, NODELIST or ALLNODES statement.

The coordinate system is implied to be that referenced in the FREEDOM statement for this node or set of nodes.

See note (1).

Example:

PRESET1 (#N12,2,0.1E-5);
prescribes a displacement of 0.1E-5 in y-direction referred to in the FREEDOM statement of node #N12.

B.4.19 PRESET2 (nodal_reference,DX,DY,DZ,DXX,DYY,DZZ);

```
PROPERTY preset2;
   DX,DY,DZ             : OPTIONAL real;
   DXX,DYY,DZZ          : OPTIONAL real;
   OF
   nodal_reference      : SELECT(gnode,nodelist,allnodes);
END_PROPERTY;

TYPE
   gnode = SELECT (node, trn);
END_TYPE;
```

Assigns prescribed displacements DX,DY,... in several directions to the nodes given in nodal_reference. This parameter may refer to a NODE, TRN, NODELIST or ALLNODES statement (this may be embedded). If one of the components is omitted, the respective degree-of-freedom is free. If one of the components is set to zero, the effect is the same as if this degree of freedom was fixed.

The coordinate system is implied to be that referenced in the FREEDOM statement for this node or set of nodes.

See note (1).

Example:

```
PRESET2 (NODELIST(#N1,#N2,#N3,#N4),1.,0.,1.,,,);
```

prescribes for the listed nodes a displacement of 1.0 in x- and z-direction. The y-direction is fixed, the rotations are free.

B.4.20 MPC (name: (list_of_nodes), (LIST_OF_DEGREES_OF_FREEDOM), (LIST_OF_COEFFICIENTS));

```
ENTITY mpc;
   list_of_nodes          : LIST [2 TO *] OF gnode;
   LIST_OF_DOF            : LIST [2 TO *] OF integer;
   LIST_OF_COEFFICIENTS : LIST [2 TO *] OF real;
END_ENTITY;
```

Enforces a linear relationship between two or more degrees of freedom.

The coordinate system is that referenced in the FREEDOM statement for each node.

Example:

```
MPC(#mpc1:(#n1,#n2),(1,2),(1,-1));
```

This statement imposes a multipoint constraint on nodes n1 and n2 such that the y displacement of n2 is equal to the x displacement of n1, i.e. $1x_{n1} - 1y_{n2} = 0$

B.4.21 TEMP (nodal_reference,TE);

```
PROPERTY temp;
   TE                    : real;
OF
   nodal_reference       : SELECT(gnode,nodelist,allnodes);
END_PROPERTY;

TYPE
   gnode = SELECT (node, trn);
END_TYPE;
```

Prescribes the temperature TE at the nodes given in nodal_reference. This parameter may refer to a NODE, TRN, NODELIST or ALLNODES statement. This is to be used only for applying a temperature constraint in thermal analysis.

See notes (2) and (4).

Example:

```
TEMP (NODELIST((#N12,#N13,#N14,#N15)),35.);
```

prescribes a temperature of 35 temperature units at the given nodes.

B.4.22 LOADP1 (name:IDIR,cosys,F);

```
ENTITY loadp1;
   IDIR                : integer;
   cosys               : OPTIONAL coord;
   F                   : real;
WHERE
   IDIR >= 1 AND IDIR <=6;
END_ENTITY;
```

Defines a static force (IDIR=1,2,3) or moment (IDIR= 4,5,6) of size F. This load can be attached to a list of nodes or treenodes via a NODELOAD statement. cosys is the name of a coordinate system.

See note (1).

Example:

```
LOADP1 (#load3: 1,#co6,25.5);
       defines a load 25.5 in x-direction of coordinate system #co6.
```

B.4.23 LOADP2 (name:FX,FY,FZ,FXX,FYY,FZZ,cosys);

```
ENTITY loadp2;
   FX,FY,FZ            : OPTIONAL real;
   FXX,FYY,FZZ         : OPTIONAL real;
   cosys               : OPTIONAL coord;
END_ENTITY;
```

Defines a load by its 6 components. Not existing components may be omitted or set zero. cosys is the name of a coordinate system.

See note (1).

Example:

```
LOADP2 (#load4: ,,,1.,1.,,);
       defines moments around x- and y-axis of the global coordinate
       system.
```

B.4.24 FREQRG (name:AM,PH);

```
ENTITY freqrg;
   AM                       : real;
   PH                       : real;
END_ENTITY;
```

Defines a sinusoidal excitation in the frequency domain:

AM	amplitude of a sinus
PH	phase of a sinus

Example:

```
FREQRG (#f1:1000.0,180.0);
```

B.4.25 STEADY (name:AM,PH,FR);

```
ENTITY steady;
   AM                       : real;
   PH                       : real;
   FR                       : real;
END_ENTITY;
```

Defines a sinusoidal excitation in the time domain for steady state response:

AM	amplitude of a sinus
PH	phase of a sinus
FR	frequency of a sinus

Example:

```
STEADY (#st1:0.2,180.0,20000.0);
```

B.4.26 IMPULS (name:AI,TI);

```
ENTITY impuls;
   AI                    : real;
   TI                    : real;
END_ENTITY;
```

Defines Dirac's impulse:

AI	amplitude of an impulse (note, that dimension is force*time or moment*time)
TI	time for the impulse

Example:

```
IMPULS (#im1:100.0,0.025);
```

B.4.27 RECTAN (name:AR,T1,T2);

```
ENTITY rectan;
   AR                    : real;
   T1                    : real;
   T2                    : real;
END_ENTITY;
```

Defines a rectangular forcing function in the time domain:

T1,T2	time interval in which the load is acting
AR	load value between T1 and T2 (outside it is zero)

Example:

```
RECTAN (#rect1:10000.0,0.05,2.0);
```

B.4.28 SAWTOO (name:AS,T1,T2);

```
ENTITY sawtoo;
   AS                      : real;
   T1                      : real;
   T2                      : real;
END_ENTITY;
```

Defines a sawtooth forcing function in the time domain:

T1,T2	time interval in which the load is acting
AS	load value at T2 (load increases linearly from T1 to T2; outside the interval it is zero)

Example:

```
SAWTOO (#saw1:125.0,0.,10.9);
```

B.4.29 SINUS (name:AM,PH,FR,T1,T2);

```
ENTITY sinus;
   AM                   : real;
   PH                   : real;
   FR                   : real;
   T1                   : real;
   T2                   : real;
END_ENTITY;
```

Defines a sinusoidal function:

AM	amplitude of a sinus
PH	phase of a sinus
FR	frequency of a sinus
T1,T2	time interval in which the load is acting

Example:

```
SINUS (#sin1:20.,90.0,1020.0,0.,3600.0);
```

B.4.30 SWEEPL (name:AM,FE,T1,T2);

```
ENTITY sweepl;
   AM                    : real;
   FE                    : real;
   T1                    : real;
   T2                    : real;
END_ENTITY;
```

Defines a linear sweep in the frequency domain:

AM	amplitude of a sinus
T1,T2	time interval in which the load is acting
FE	frequency at T2 (frequency increases from T1 to T2)

Example:

```
SWEEPL (#1s1:0.9,20000.0,10.0,34.0);
```

B.4.31 SWEEPE (name:AM,TC,FE,T1,T2);

```
ENTITY sweepe;
   AM                    : real;
   TC                    : real;
   FE                    : real;
   T1                    : real;
   T2                    : real;
END_ENTITY;
```

Defines an exponential sweep in the frequency domain:

AM	amplitude of a sinus
T1,T2	time interval in which the load is acting
FE	frequency at T2 (frequency increases from T1 to T2)
TC	time constant for exponential sweep or starting; at time T1+TC the frequency has reached 1-1/e (approximately 63.21%) of the maximum value FE

Example:

```
SWEEPE (#sel:100.,8.3,666.,0.0,10.0);
```

B.4.32 STARTL(name:UB,FE,T1,T2);

```
ENTITY startl;
   UB                    : real;
   FE                    : real;
   T1                    : real;
   T2                    : real;
END_ENTITY;
```

Defines a linearly starting unbalance force:

```
   T1,T2  time interval in which the load is acting
   FE     frequency at T2 (frequency increases from T1 to T2)
   UB     unbalance m*r (m=mass, r=radius),
          the amplitude is  UB*(2*pi*F)^2)  with 0<=F<=FE
```

Example:

```
STARTL (#stl:234.0,20.,0.0,99.0);
```

B.4.33 STARTE (name:UB,TC,FE,T1,T2);

```
ENTITY starte;
   UB                  : real;
   TC                  : real;
   FE                  : real;
   T1                  : real;
   T2                  : real;
END_ENTITY;
```

Defines an exponentially starting unbalance force:

```
   T1,T2  time interval in which the load is acting
   FE     frequency at T2 (frequency increases from T1 to T2)
   TC     time constant for exponential sweep or starting;
          at time T1+TC the frequency has reached 1-1/e
          (approximately .6321) of the maximum value FE
   UB     unbalance m*r (m=mass, r=radius),
          the amplitude is  UB*(2*pi*F)^2)  with 0<=F<=FE
```

Example:

```
STARTE (#ste:234.0,70.,20.,0.0,99.0);
```

B.4.34 RANDOM (name:AP,SO,T1,T2);

```
ENTITY random;
   AP                    : real;
   SO                    : real;
   T1                    : real;
   T2                    : real;
END_ENTITY;
```

Defines random noise excitation:

AP	peak amplitude
T1,T2	time interval in which the load is acting
SO	starting value for random number generator; equal SO in different random loads will produce the same random excitations.

Example;

```
RANDOM (#rnd:10.8,2.0,0.0,120.0);
```

B.4.35 DYNLOAD (name:IDIR,cosys,dyn_subkw);

```
ENTITY dynload;
   IDIR                 : integer;
   cosys                : coord;
   dyn_subkw            : SELECT ( freqrg, steady, impuls, rectan,
                                   sawtoo, sinus, sweepl, sweepe,
                                   startl, starte, random );
WHERE
   IDIR >= 1 AND IDIR <= 6;
END_ENTITY;
```

Defines a dynamic load in direction IDIR (of -if specified- the coordinate system cosys). Size and time dependence are referenced in by dyn_subkw.

See notes (1) and (3).

Example:

```
DYNLOAD (#dyn1:3,#coor1,#im2);
       defines a dynamic load in the z-direction of coordinate system
       #coor1, the type of load is referenced by #im2.
```

B.4.36 DISCRLOAD (name:IDIR,cosys,(T1,T2,T3,...,TN),(V1,V2,V3,...,VN));

```
ENTITY discrload;
   IDIR                 : integer;
   cosys                : OPTIONAL coord;
   TIMES                : LIST [1 TO *] OF real;
   VALUES               : LIST [1 TO *] OF real;
WHERE
   IDIR >= 1 AND IDIR <= 6;
   SIZEOF ( TIMES ) = SIZEOF ( VALUES );
END_ENTITY;
```

Defines a dynamic load given in N discrete time steps, between these steps linear interpolation is valid.

Both DYNLOAD and DISCRLOAD loads can be attached to a node (and a load case) via the NODELOAD statement. For dynamic response analyses either a frequency range or a time range is required.

See notes (1) and (3).

Examples:

```
DISCRLOAD (#loadD:3,,(0.0,0.1,0.2,0.3,0.8,0.9,1.0),
          (0.0,75.,95.,100.,100., 80.,0.0));
```

describes a transient load in 7 time steps beginning and ending with 0, with a peak value of 100 lasting from 0.3 to 0.8.

B.4.37 NODELOAD (load_name,LCASE,nodal_reference);

```
LINK nodeload;
   LCASE                  : integer;
OF
   load_name              : SELECT ( loadpl,loadp2,dynload,discrload );
   nodal_reference        : SELECT ( gnode, nodelist, allnodes);
END_LINK;

TYPE
   gnode = SELECT (node, trn);
```

Assigns a point load for the loadcase number, LCASE, to the nodes given in nodal reference. This parameter may refer to a NODE, TRN, NODELIST or ALLNODES statement.

Example:

```
NODELOAD (#load3,1,#list3);
       assigns load #load3 to a previously defined list of nodes (#list3),
       valid for loadcase 1.
```

B.4.38 LOADL1 (name:IDIR,cosys,FL);

```
ENTITY loadl1;
   IDIR                    : integer;
   cosys                   : OPTIONAL coord;
   FL                      : real;
WHERE
   IDIR >= 1 AND IDIR <=6;
END_ENTITY;
```

Defines a line load of size FL (load per unit length) in direction IDIR (of the given coordinate system if specified).

See note (1).

Example:

LOADL1 (#load5: 1,#co6, 25.5);
defines a line load 25.5 in x-direction of coordinate system #co6.

B.4.39 LOADL2 (name:FLX,FLY,FLZ,FLXX,FLYY,FLZZ,cosys);

```
ENTITY loadl2;
   FLX,FLY,FLZ             : OPTIONAL real;
   FLXX,FLYY               : OPTIONAL real;
   FLZZ                    : OPTIONAL real;
   cosys                   : OPTIONAL coord;
END_ENTITY;
```

Defines a line load with components FLZ, FLY, ..(load per unit length).

See note (1).

Example:

LOADL2 (#load4: 1.1,1.2,1.3,,,,#co3);
defines a line load with components 1.1, 1.2, 1.3 in x-, y-, z-direction of coordinate system #co3.

B.4.40 EDGELOAD (load_name,LCASE,element,p1,p2);

```
LINK edgeload;
   LCASE                : integer;
OF
   load_name            : SELECT ( loadl1, loadl2 );
   element              : gelement;
   p1,p2                : gnode;
END_LINK;

TYPE
   gnode = SELECT (node, trn);
   gelement = SELECT (element, tre);
   element = SELECT ( bar, beam2, spring, damper, mass, tri3,
                        tri6, tri7v, quad4, quad8, quad9v, tetra4,
                        tetra10, tetra15v, penta6, penta15,
                        penta21v, hexa8, hexa20, hexa21v );
END_TYPE;
```

Attaches a load to an edge of a 2D or 3D element. The edge is defined by p1 and p2. The element may be the name of an element or a tree element and the nodes may be the names of nodes or tree nodes.

See note (2).

Example:

EDGELOAD (#load3,2,#E9,#N17,#N18);
assigns load #load3 for load case 2 to an edge of element #E9 given by the two nodes #N17 and #N18.

B.4.41 LOADS1 (name:IDIR,cosys,FS);

```
ENTITY loads1;
   IDIR                    : integer;
   cosys                   : OPTIONAL coord;
   FS                      : real;
WHERE
   IDIR >= 1 AND IDIR <=6;
END_ENTITY;
```

Defines a surface load of size FS (pressure) in direction IDIR.

See note (1).

Example:

LOADS1 (#load3: 1,#co6, 25.5);
defines a surface load 25.5 in x-direction of coordinate system #co6.

B.4.42 LOADS2 (name:FSX,FSY,FSZ,FSXX,FSYY,FSZZ,cosys);

```
ENTITY loads2;
   FSX,FSY,FSZ             : OPTIONAL real;
   FSXX, FSYY              : OPTIONAL real;
   FSZZ                    : OPTIONAL real;
   cosys                   : OPTIONAL coord;
END_ENTITY;
```

Defines a surface load with components FSX,FSY,... (load per unit area).

See note (1).

Example:

LOADS2 (#load4: 1.1,1.2,1.3,,,,#co3);
defines a surface load with components 1.1, 1.2, 1.3 in x-, y-, z-direction of coordinate system #co3.

B.4.43 SURFLOAD (load_name,LCASE,element,p1,p2,p3,p4);

```
LINK surfload;
   LCASE                : integer;
OF
   load_name            : SELECT ( loads1, loads2 );
   element              : gelement;
   p1,p2,p3,            : gnode;
   p4                   : OPTIONAL gnode;
END_LINK;

TYPE
   gnode = SELECT (node, trn);
   gelement = SELECT ( element, tre );
   element = SELECT ( bar, beam2, spring, damper, mass, tri3,
                      tri6, tri7v, quad4, quad8, quad9v, tetra4,
                      tetra10, tetra15v, penta6, penta15,
                      penta21v, hexa8, hexa20, hexa21v );
END_TYPE;
```

Attaches a load to a surface of a 3D-element. The surface is defined by all of its vertices; p1, p2, p3, p4. p4 is omitted if the surface is triangular. The element and nodes may also be tree element and tree nodes.

See note (2).

Example:

```
SURFLOAD (#load3,1,#E9,#N17,#N18,#N19,);
        assigns load #load3 for load case 1 to a triangular surface of
        element #E9 given by the three nodes #N16, #N17 and #N18.
```

B.4.44 LOADV1 (name:IDIR,cosys,FV);

```
ENTITY loadv1;
   IDIR                  : integer;
   cosys                 : OPTIONAL coord;
   FV                    : real;
WHERE
   IDIR >= 1 AND IDIR <=6;
END_ENTITY;
```

Defines a volume load of size FV (load per unit volume) in direction IDIR.

See note (1).

Example:

LOADV1 (#load3: 1,, 25.5);
defines a volume load 25.5 in x-direction of the global coordinate system.

B.4.45 LOADV2 (name:FVX,FVY,FVZ,FVXX,FVYY,FVZZ,cosys);

```
ENTITY loadv2;
   FVX,FVY,FVZ           : OPTIONAL real;
   FVXX, FVYY            : OPTIONAL real;
   FVZZ                  : OPTIONAL real;
   cosys                 : OPTIONAL coord;
END_ENTITY;
```

Defines a line load with components FVX,FVY,... (load per unit volume).

See note (1).

Example:

LOADV2 (#load4: 1.1,1.2,1.3,,,,#co3);
defines a volume load with components 1.1, 1.2, 1.3 in x-, y-, z-direction of coordinate system #co3.

B.4.46 ELEMLOAD (load_name,LCASE,element_reference);

```
LINK elemload;
   LCASE                : integer;
OF
   load_name            : SELECT ( loadl1, loadl2, loadsl, loads2,
                                   loadvl, loadv2 );
   element_reference    : SELECT(gelement, elemlist, allelems);
END_LINK;

TYPE
   gelement = SELECT ( element, tre );
   element = SELECT ( bar, beam2, spring, damper, mass, tri3,
                      tri6, tri7v, quad4, quad8, quad9v, tetra4,
                      tetra10, tetra15v, penta6, penta15,
                      penta21v, hexa8, hexa20, hexa21v );
END_TYPE;
```

Assigns a distributed load (line, surface or volume) to one or more elements of the same dimension as the load; i.e. line loads can be attached to 1D elements (e.g. BEAM2), surface loads can be attached to 2D elements (e.g. QUAD8), volume loads can be attached to 3D elements (e.g. HEXA8).

element_reference may refer to the name of an element or a tree element, the name of an elemlist or the name of an allelems which assigns the load to all the elements in a list of objects. Any of these keywords may be embedded.

See note (2).

Example:

```
ELEMLOAD (#load3,2,#list99);
       assigns load #load3 to the elements of list #list99 for loadcase 2.
```

B.4.47 GRAVLOAD (name:LCASE,IDIR,cosys,ACCELERATION);

```
ENTITY gravload;
   LCASE                 : integer;
   IDIR                  : integer;
   cosys                 : OPTIONAL coord;
   ACCELERATION          : real;
WHERE
   IDIR >= 1 AND IDIR <= 6;
END_ENTITY;
```

Defines a gravity load in direction IDIR with a loadcase LCASE. This load is applied to the whole object referred to by the analysis statement where this loadcase number is used. The coordinate system may be omitted.

See note (1).

Example:

```
GRAVLOAD (#load7;100,3,,-9.81);
       defines a gravity load of 9.81 in the -z-direction.
```

B.4.48 TEMPLOAD (nodal_reference,LCASE,TE);

```
PROPERTY tempload;
   LCASE                 : integer;
   TE                    : real;
OF
   nodal_reference       : SELECT(gnode,nodelist,allnodes);
END_PROPERTY;

TYPE
   gnode = SELECT (node, trn);
END_TYPE;
```

Prescribes for the loadcase LCASE the temperature TE at the nodes given in nodal_reference. This parameter may refer to a NODE, TRN, NODELIST or ALLNODES statement. This is used only for static analysis with thermal loads.

See note (4).

Example:

```
TEMPLOAD (NODELIST((#N12,#N13,#N14,#N15)),2,35.);
```

prescribes a temperature of 35 temperature units for loadcase 2 at the given nodes.

B.4.49 CONVEC (element,LCASE,H,AMBTE,(list_of_nodes));

```
PROPERTY convec;
   LCASE                : integer;
   H                    : real;
   AMBTE                : real;
   list_of_nodes        : LIST [ 1 TO 4 ] OF gnode;
OF
   element              : gelement;
END_PROPERTY;

TYPE
   gnode = SELECT (node, trn);
   gelement = SELECT ( element, tre );
   element = SELECT ( bar, beam2, spring, damper, mass, tri3,
                        tri6, tri7v, quad4, quad8, quad9v, tetra4,
                        tetra10, tetra15v, penta6, penta15,
                        penta21v, hexa8, hexa20, hexa21v );
END_TYPE;
```

Specifies heat convection through one surface of a certain element or tree element.

LCASE = load case.
H = convection heat transfer coefficient.
AMBTE = ambient temperature.
list_of_nodes = a list of 1 to 4 node names or tree node names which specify the surface.

Note that the meaning of the word 'surface' here is slightly different from the surface mentioned in connection with static loads. Here it is possible to describe an edge of a shell element as a surface (with the area edge-length times element thickness).

Example:

CONVEC (#E15,2,1.56E-5,20.,(#N23,#N25));
specifies heat convection at one surface (this is here an edge of a 2D element) of element #E15 for loadcase 2.

B.4.50 FLUX (element,LCASE,HEAT_FLUX,(list_of_nodes));

```
PROPERTY flux;
   LCASE                : integer;
   HEAT_FLUX            : real;
   list_of_nodes        : LIST[ 1 TO 4 ] OF gnode;
OF
   element              : gelement;
END_PROPERTY;

TYPE
   gnode = SELECT (node, trn);
   gelement = SELECT ( element, tre );
   element = SELECT ( bar, beam2, spring, damper, mass, tri3,
                      tri6, tri7v, quad4, quad8, quad9v, tetra4,
                      tetra10, tetra15v, penta6, penta15,
                      penta21v, hexa8, hexa20, hexa21v );
END_TYPE;
```

Specifies a prescribed heat flux through one surface of a certain element or tree element. The parameters LCASE and list_of_nodes are the same as in the CONVEC statement.

Example:

```
FLUX (#E15,2,1.56E-5,(#N23,#N25,#N27,#N29));
       specifies heat flux through one surface of element #E15 for
       loadcase 2.
```

B.4.51 QVOL (element_reference,LCASE,HPV);

```
PROPERTY qvol;
   LCASE                 : integer;
   HPV                   : real;
OF
   element_reference     : SELECT (gelement,elemlist,allelems);
END_PROPERTY;

TYPE
   gelement = SELECT ( element, tre );
   element = SELECT ( bar, beam2, spring, damper, mass, tri3,
                      tri6, tri7v, quad4, quad8, quad9v, tetra4,
                      tetra10, tetra15v, penta6, penta15,
                      penta21v, hexa8, hexa20, hexa21v );
END_TYPE;
```

Establishes a heat source, for load case LCASE, distributed over the elements or tree elements given in element_reference. This parameter may refer to an ELEMENT, TRE, ELEMLIST or ALLELEMS statement.

HPV = heat generation per unit volume

Example:

```
QVOL (ALLELEMS((#OB1)),2,4.5);
       specifies heat production in all elements of object #OB1 for
       loadcase 2.
```

B.4.52 QVOLP (nodal_reference,LCASE,HEAT);

```
PROPERTY qvolp;
   LCASE                 : integer;
   HEAT                  : real;
OF
   nodal_reference       : SELECT (gnode,nodelist,allnodes);
END_PROPERTY;

TYPE
   gnode = SELECT ( node, trn );

END_TYPE;
```

Establishes a point heat source, for load case LCASE, at the nodes or tree nodes given in nodal_reference. This parameter may refer to a NODE, TRN, NODELIST or ALLNODES statement.

HEAT = rate of heat generation

Example:

```
QVOLP (#n4,4,0.5);
       specifies heat production at node n4 for loadcase 4.
```

B.4.53 FREQRANGE (name:F1,DF,F2);

```
ENTITY freqrange;
   F1                     : real;
   DF                     : OPTIONAL real;
   F2                     : real;
END_ENTITY;
```

Defines a frequency range which is referenced by an analysis statement. The frequency steps are F1, F1+DF, F1+2*DF, ... so far as F2 (maximum frequency) is not exceeded. If the stepwidth (DF) is omitted then the analysis program should select a suitable frequency step.

Example:

```
FREQRANGE (#ra1: 0., 1., 250.);
```

defines a frequency range from 0...250 in steps of 1 frequency unit.

B.4.54 TIMERANGE (name:T1,DT,T2);

```
ENTITY timerange;
   T1                 : real;
   DT                 : OPTIONAL real;
   T2                 : real;
END_ENTITY;
```

Defines a time range which is referenced by an analysis statement. The time steps are T1, T1+DT, T1+2*DT, ... so far as T2 (end time) is not exceeded. If the stepwidth (DT) is omitted then the analysis program should select a suitable time step.

Example:

```
TIMERANGE (#ra2: 0., 1., 250.);
       defines a time range from 0...250 in steps of 1 time unit.
```

B.4.55 MDCONST (md_name:D);

```
ENTITY mdconst;
   D                    : real;
END_ENTITY;
```

Specifies the value for modal damping which can be used instead of structural damping. The value D is the critical damping ratio and is valid for all modes.

Example:

```
MDCONST (#damp, 0.05);
       specifies a modal damping of 5 % of the critical damping constantly
       for all modes.
```

B.4.56 MODALDAMP(modaldamp_name:mddamp_spec);

```
ENTITY modaldamp;
   mddamp_spec          : SELECT (mdconst);
END_ENTITY
```

Defines modal damping which can be used instead of structural damping. The damping itself is defined in a MDCONST statement.

Examples:

```
MODALDAMP (#modamp, #con);
       defines a modal damping which is specified in #con.

MODALDAMP (#modamp, MDCONST (0.05) );
       defines a modal damping of 5 % of the critical damping constantly
       for all modes. It can be referenced by analysis statements.
```

B.4.57 ANSTATIC (ana-id:object,(LIST_OF_LOADCASES));

```
ENTITY anstatic;
   object               : object;
   LIST_OF_LOADCASES    : SET OF integer;
END_ENTITY;

TYPE
   object = SELECT (collect, assem, scond, copy);
END_TYPE;
```

Performs a static analysis on an object. The object must have been defined in a collect statement or must have been derived from such an object (e.g. via a condensation).

LIST_OF_LOADCASES is a sequence of integer numbers denoting the loadcases to be computed.

The ANSTATIC analysis is also valid in the case of thermal loads.

The name ana-id is to be referenced by result data.

Example:

```
ANSTATIC (#anal:#OB1,(1,2,5,6));
        performs a static analysis on object #OB1 with loadcases 1,2,5 and
        6.
```

B.4.58 ANREIGV (ana-id:object);

```
ENTITY anreigv;
   object              : object;
END_ENTITY;

TYPE
   object = SELECT (collect, assem, scond, copy);
END_TYPE;
```

Performs a real eigenvalue analysis on an object. The object must have been defined in a collect statement or must have been derived from such an object (e.g. via a condensation or a copy).

Example:

```
ANREIGV (#ana2: #OB2);
```

B.4.59 ANCEIGV(anceigv-name:object,ANCE_TYPE,dcond,modaldamp);

```
ENTITY anceigv;
   object              : object;
   ANCE_TYPE           : ENUMERATION OF (PHYS, REIG);
   dcondname           : OPTIONAL dcond;
   dampname            : OPTIONAL modaldamp;
END_ENTITY;

TYPE
   object = SELECT (collect, assem, scond, copy);
END_TYPE;
```

Performs a complex eigenvalue analysis on an object. The type indicates the computation method:

```
       PHYS = analysis with physical matrices
       REIG = analysis with matrices based on real eigenvectors
              (for REIG a DCOND step is needed)
```

If REIG is specified, then the eigenvalue analysis is performed on the object produced by the dynamic condensation pointed to by dcond.

modaldamp points to a modal damping specification. If present, the structural damping matrix will be disabled.

Example:

```
ANCEIGV (#ana5: #OB2,REIG,#OB5,);
```

performs a complex eigenvalue analysis on the object #OB2. Modal matrices (based on the dynamic condensation producing #OB5) and structural damping are used.

B.4.60 ANFRESP(anfresp_name:object,ANFR_TYPE,dcond,freqrange,modaldamp, (LIST_OF_LOADCASES));

```
ENTITY anfresp;
   object               : object;
   ANFR_TYPE            : ENUMERATION OF (PHYS, REIG, CEIG);
   dcondname            : OPTIONAL dcond;
   freqrange            : freqrange;
   dampname             : OPTIONAL modaldamp;
   LIST_OF_LOADCASES    : SET OF integer;
END_ENTITY;

TYPE
   object = SELECT (collect, assem, scond, copy);
END_TYPE;
```

Performs a frequency response analysis on an object. The type indicates the computation method:

```
       PHYS = analysis with physical matrices
       REIG = analysis with matrices based on real eigenvectors
       CEIG = analysis with matrices based on complex eigenvectors
              (for REIG and CEIG a DCOND step is needed)
```

If REIG or CEIG is specified the frequency response is performed on the object produced by the dynamic condensation pointed to by dcond. If PHYS is specified DCOND is omitted.

The frequency range must be previously defined.

LIST_OF_LOADCASES is a sequence of integer numbers denoting the loadcases to be computed.

modaldamp points to a modal damping specification. If present, the structural damping matrix will be disabled.

Examples:

```
ANFRESP (#ana3: #OB2,PHYS,,#ral,,(1));
```

performs a response analysis with one loadcase in the frequency range #ral on the object #OB2. Physical matrices and structural damping are used.

```
ANFRESP (#ana3: #OB2,REIG,#OB5,#ral,#modamp,(1));
```

performs a response analysis with one loadcase in the frequency range #ral on the object #OB2. Modal matrices (obtained from dynamic condensation in object #OB5) are used. Structural damping is disabled and modal damping is used instead.

B.4.61 ANTRESP(antresp_name:object,ANTR_TYPE,dcond,timerange,modaldamp, (LIST_OF_LOADCASES));

```
ENTITY antresp;
   object               : object;
   ANTR_TYPE            : ENUMERATION OF (PHYS, REIG, CEIG);
   dcondname            : OPTIONAL dcond;
   timerange            : timerange;
   dampname             : OPTIONAL modaldamp;
   LIST_OF_LOADCASES    : SET OF integer;
END_ENTITY;

TYPE
   object = SELECT (collect, assem, scond, copy);
END_TYPE;
```

Performs a transient response analysis of an object. The type indicates the computation method:

PHYS = analysis with physical matrices
REIG = analysis with matrices based on real eigenvectors
CEIG = analysis with matrices based on complex eigenvectors
(for REIG and CEIG a DCOND step is needed)

If REIG or CEIG is specified, the transient response analysis is performed on the object produced by the dynamic condensation pointed to by dcond. If PHYS is specified dcond is omitted.

The timerange must be previously defined.

LIST_OF_LOADCASES is a sequence of integer numbers denoting the loadcases to be computed.

modaldamp points to a modal damping specification. If present, the structural damping matrix will be disabled.

Example:

ANTRESP (#ana4: #OB2,REIG,#OB5,#ra2,,(1,2));
performs a response analysis with two loadcases in the time range #ra2 on the object #OB2. Modal matrices (based on the dynamic condensation producing #OB5) and structural damping are used.

B.4.62 ANSTEMP (ana-id:object,(LIST_OF_LOADCASES));

```
ENTITY anstemp;
   object               : object;
   LIST_OF_LOADCASES    : SET OF integer;
END_ENTITY;

TYPE
   object = SELECT (collect, assem, scond, copy);
END_TYPE;
```

Performs a linear thermal analysis on an object.

LIST_OF_LOADCASES is a list of integers specifying the loadcases to be computed.

See note (4).

Example:

```
ANSTEMP (#ana6: #OB3, (1,2) );
       performs a thermal analysis with two loadcases on object #OB3.
```

B.4.63 OUTDISP (ana-id,nodal_reference,cosys);

```
PROPERTY outdisp;
   nodal_reference       : SELECT(gnode,nodelist,allnodes);
   cosys                 : OPTIONAL coord;
OF
   ana_id                : SELECT (anstatic, anreigv, anfresp, antresp,
                                        anceigv);
END_PROPERTY;

TYPE
   gnode = SELECT (node, trn);
END_TYPE;
```

Requests output of displacements for the node or nodes given in nodal reference. This parameter may refer to a NODE, TRN, NODELIST or ALLNODES statement.

See note (5).

Example:

OUTDISP (#anal,#listl,#col);
requests displacements output for analysis #anal in coordinate system #col. #listl was defined in a NODELIST or ALLNODES statement.

B.4.64 OUTSTRESS (ana-id,OUTS_SUBKW);

```
PROPERTY outstress;
   OUTS_SUBKW           : SELECT (NODAL,ELEMENTAL );
OF
   ana_id               : SELECT (anstatic, anreigv, anfresp, antresp,
                                  anceigv);
END_PROPERTY;
```

Requests output of nodal or elemental stresses for nodes (subkeyword NODAL) or elements (subkeyword ELEMENTAL).

ana-id is the name defined in an analysis (e.g. ANSTATIC) command.

See note (1) and (5).

Example:

```
OUTSTRESS (#ana2,#a5);
       requests stress output from analysis #ana2 at nodes or elements
       referenced by #a5.
```

B.4.65 OUTSTRAIN (ana-id,outs_subkw);

```
PROPERTY outstrain;
   outs_subkw            : SELECT (NODAL,ELEMENTAL );
OF
   ana_id                : SELECT (anstatic, anreigv, anfresp, antresp,
                                   anceigv);
END_PROPERTY;
```

Requests output of strains for certain nodes or elements. The parameters are the same as for OUTSTRESS.

See notes (1) and (5).

Example:

OUTSTRAIN (#ana2,#all2);
requests strain output from analysis #ana2 for nodes or elements referenced by #all2.

B.4.66 OUTFORCE (ana-id,ITYP,nodal_reference,cosys);

```
PROPERTY outforce;
   TYPE_CODE              : INTEGER;
   nodal_reference        : SELECT ( gnode,nodelist,allnodes );
   cosys                  : OPTIONAL coord;
OF
   ana_id                 : SELECT (anstatic, anreigv, anfresp, antresp,
                                    anceigv)
WHERE;
   TYPE_CODE >= 1 AND TYPE_CODE <= 3;
END_PROPERTY;

TYPE
   gnode = SELECT (node, trn);
END_TYPE;
```

Requests output of spring forces (ITYP = 1), damping forces (ITYP = 2) or inertia forces (ITYP = 3) for the nodes given by nodal reference. This parameter may refer to a NODE, TRN, NODELIST or ALLNODES statement. (Any of these can be embedded.)

The optional cosys defines the coordinate system in which the results are to be output.

See notes (1) and (5).

Example:

```
OUTFORCE (#ana5,3,#N25,);
       outputs inertia forces of analysis #ana5 for node #N25 in global
       coordinates.
```

B.4.67 FIRSTFR (name:N);

```
ENTITY firstfr;
   N                     : integer;
END_ENTITY;
```

Used in OUTEIGFR keyword to request output of the first N eigenfrequencies.

Example;

```
FIRSTFR (#f30:30);
       asks for the first 30 eigenfrequencies.
```

B.4.68 MODELIST (name:MODELIST);

```
ENTITY modelist;
   MODELIST              : SET OF integer;
END_ENTITY;
```

Used in OUTEIGFR keyword to request output of eigenfrequencies corresponding to the modes listed.

Example:

```
MODELIST (#mod7:(1,3,5,7));
       asks for the eigenfrequencies of modes 1, 3, 5 and 7.
```

B.4.69 FREQOUT (name:FIRSTFREQ,SECONDFREQ);

```
ENTITY freqout;
   FIRSTFREQ             : real;
   SECONDFREQ            : real;
END_ENTITY;
```

Used in OUTEIGFR keyword to request output of eigenfrequencies between the stated frequencies.

Example:

```
FREQOUT (#man:2000.0,20000.0);
       asks for all eigenfrequencies between 2000 and 20000 frequency
       units.
```

B.4.70 OUTEIGFR (ana-id,outeig_subkw);

```
PROPERTY outeigfr;
   outeig_subkw          : SELECT (firstfr,modelist,freqout);
OF
   ana_id                : SELECT (anreigv,anceigv);
END_PROPERTY;
```

Requests output of the computed eigenfrequencies.

If the analysis is ANCEIGV, i.e. complex, then the critical damping ratios will also be output for the modes requested.

See note (5).

Example:

```
OUTEIGFR (#ana2,#ffr30);
```

B.4.71 OUTMODESH (ana-id,(LIST_OF_MODES));

```
PROPERTY outmodesh;
   LIST_OF_MODES        : SET OF integer;
OF
   ana_id               : SELECT (anreigv,anceigv);
END_PROPERTY;
```

Requests output of the eigenvectors (modes) whose numbers are given in LIST_OF_MODES.

See note (5).

Examples:

```
OUTMODESH (ana2,(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17));
       asks for the output of the first 17 modeshapes.
```

B.4.72 OUTTEMP (ana-id,nodal_reference);

```
PROPERTY outtemp;
   nodal_reference        : SELECT ( gnode,nodelist,allnodes )
OF
   ana_id                 : anstemp;
END_PROPERTY;

TYPE
   gnode = SELECT (node, trn);
END_TYPE;
```

Requests output of temperature(s) for the node(s) given by nodal reference. This parameter may refer to a node or to a TRN, NODELIST or ALLNODES statement (these can be embedded).

See note (5).

Example:

```
OUTTEMP (#ana6,ALLNODES((#OB1)));
       outputs temperatures of analysis #ana6 for all nodes of object
       #OB1.
```

B.4.73 OUTFLUX (ana-id,element_reference);

```
PROPERTY outflux;
   element_reference     : SELECT (gelement,elemlist,allelems );
OF
   ana_id                : anstemp
END-PROPERTY;

TYPE
   gelement = SELECT ( element, tre );
   element = SELECT ( bar, beam2, spring, damper, mass, tri3,
                      tri6, tri7v, quad4, quad8, quad9v, tetra4,
                      tetra10, tetra15v, penta6, penta15,
                      penta21v, hexa8, hexa20, hexa21v );
END_TYPE;
```

Requests output of heat flux over element boundaries.

The elements are those given in the element_reference, which may refer to an element statement or to a TRE, ELEMLIST or ALLELEMS statement (any of these may be embedded).

See note (5).

Examples:

```
OUTFLUX (#ana6,#E123);
       asks for flux output for element #E123.

OUTFLUX (#ana6,#all1);
       asks for flux output for all elements of objects mentioned in the
       ALLELEMS statement defining #all1.
```

B.4.74 CLOSEANA (analysis_name);

```
PROPERTY closeana;
OF
   analysis_name          : openana;
END_PROPERTY;
```

Closes this analysis block.

Examples:

```
CLOSEANA (#ANALY1);
```

B.5 Keywords for FE Results

B.5.1 OPENFERES (set_name:analysis_name,'TEXT_DESCRIPTION');

```
ENTITY openferes;
   analysis_name        :  openana;
   TEXT                 :  OPTIONAL string;
END_ENTITY;
```

Opens one block of finite element results. Each block is related to one load case number or mode number of a particular analysis. The set_name is a name for this special set of results (e.g. displacements or stresses of one load case).

The analysis_name was defined in the OPENANA block. The text can give further information about the analysis.

Example:

```
OPENFERES (#RES1: #ANALY1,'the second load case of a static analysis');
```

B.5.2 ANCASE (anstep_id,ancase_type,LCASE,STEP);

```
PROPERTY ancase;
   ancase_type          :  ENUMERATION OF (STATA, REIGA, CEIGA,
                                       FRESA, TRESA, TEMPA) ;
   LCASE                :  OPTIONAL integer;
   STEP                 :  OPTIONAL real;
OF
   anstep_id            :  SELECT (anstatic, anreigv,anfresp,
                                       antresp,anceigv, anstemp);
END_PROPERTY;
```

Defines from which analysis step (anstep_id) the results are presented. The subkeyword ancase_type explains the origin of the results, i.e.:

STATA	denotes a static analysis
REIGA	denotes a real eigenvalue analysis
CEIGA	denotes a complex eigenvalue analysis
FRESA	denotes a frequency response analysis
TRESA	denotes a time (transient) response analysis
TEMPA	denotes a temperature analysis

The list is extendible.

The parameter LCASE denotes the load case number. If the output is the mode shapes from an eigenvalue analysis then LCASE is the corresponding mode number. If the output is eigenfrequencies then the LCASE parameter is omitted. For a response analysis STEP denotes time or frequency step.

Examples:

ANCASE (#ana3,STATA,1,);
indicates loadcase 1 of static analysis #ana3.

ANCASE (#ana4,REIGA,5,);
indicates mode 5 of eigen analysis #ana4.

ANCASE (#ana5,FRESA,1,65.425);
indicates frequency step 65.425 for loadcase 1 of frequency response analysis #ana5.

B.5.3 DISPR (node_name,VX,VY,VZ,RX,RY,RZ,cosys);

```
PROPERTY dispr;
   VX,VY,VZ            :  OPTIONAL real;
   RX,RY,RZ            :  OPTIONAL real;
   cosys               :  OPTIONAL coord;
OF
   node_name           :  SELECT (node,trn);
END_PROPERTY;
```

Outputs the translations VX,VY,VZ and the rotations RX,RY,RZ (real). E.g. from a static analysis. They are calculated at the node named node_name and given in the coordinate system named cosys (default allowed). Omitted values are zero.

Example:

```
DISPR (#N17, 1.2345E-4, 2.3456E-4, 3.4567E-4,,,, );
```

gives 3 translations and three rotations (default=0.0), in global coordinates for node #N17.

B.5.4 DISPC (node_name,VX,VY,VZ,RX,RY,RZ,PVX,PVY,PVZ,PRX,PRY,PRZ,cosys);

```
PROPERTY dispc;
   VX,VY,VZ              :  OPTIONAL real;
   RX,RY,RZ              :  OPTIONAL real;
   PVX,PVY,PVZ           :  OPTIONAL real;
   PRX,PRY,PRZ           :  OPTIONAL real;
   cosys                 :  OPTIONAL coord;
OF
   node_name             :  SELECT (node,trn);
END_PROPERTY;
```

Outputs the translations VX, VY, VZ and the rotations RX, RY, RZ as amplitudes and the corresponding phases PVX, PVY, PVZ and PRX, PRY, PRZ, e.g. from a static analysis. They are calculated at the node named node_name and given in the coordinate system named cosys (default allowed). Omitted values are zero.

Example:

```
DISPC (#N17,1.0,2.0,3.0,0.4,0.5,0.6,22.2,23.5,22.4,19.7,21.9,22.0,#cosys3);
       outputs all displacements of node #N17 in coordinate system cosys3.
```

B.5.5 STRENOR (node_name,SX,SY,SZ,SXY,SXZ,SYZ,cosys);

```
PROPERTY strenor;
   SX,SY,SZ             :  OPTIONAL real;
   SXY,SXZ,SYZ          :  OPTIONAL real;
   cosys                :  OPTIONAL coord;
OF
   node_name            :  SELECT(node,trn);
END_PROPERTY;
```

Outputs the 6 nodal stress components SX,...,SYZ. They are calculated at the node named node_name and given in the coordinate system named cosys (default allowed). Omitted values are zero.

Example:

```
STRENOR (#N17, 3.523E2, -1.263E2, 5.674E1, 6.599E1, 2.162E1, 7.773E0, ,);
       outputs 6 real stress components in global coordinates for node
       #N17.
```

B.5.6 STRENOC (node_name,SX,SY,SZ,SXY,SXZ,SYZ,PSX,PSY,PSZ, PSXY,PSXZ,PSYZ,cosys);

```
PROPERTY strenoc;
   SX,SY,SZ               :  OPTIONAL real;
   SXY,SXZ,SYZ            :  OPTIONAL real;
   PSX,PSY,PSZ            :  OPTIONAL real;
   PSXY,PSXZ,PSYZ         :  OPTIONAL real;
   cosys                  :  OPTIONAL coord;
OF
   node_name              :  SELECT(node,trn);
END_PROPERTY;
```

Outputs the 6 nodal stress components SX,...,SYZ as amplitudes and the corresponding phases PSX,...,PSYZ. They are calculated at the node named node_name and given in the coordinate system named cosys (default allowed). Omitted values are zero.

Example:

```
STRENOC (#N17, 3.523E2,-1.263E2, 5.674E1, 6.599E1, 2.162E1,
         7.773E0, 123.2, 97.5, 110.3, 99.5, 82.3, 101.1 ,);
       outputs 6 complex stress components (phases in degrees) in global
       coordinates for node #N17.
```

B.5.7 STREEQR (node_name,SE);

```
PROPERTY streeqr;
   SE                    :  real;
OF
   node_name             :  SELECT(node,trn);
END_PROPERTY;
```

Outputs the equivalent stress (real) SE for the node named node_name.

Example:

```
STREEQR (#N19, 56.74);
       outputs equivalent stress for node #N19.
```

B.5.8 STREEQC (node_name,SE,PSE);

```
PROPERTY streeqc;
   SE                    :  real;
   PSE                   :  real;
OF
   node_name             :  node;
END_PROPERTY;
```

Outputs the equivalent stress (complex) SE as the amplitude and the corresponding phase PSE for the node named node_name.

Example:

```
STREEQC (#N19, 56.74, 1.25);
       outputs equivalent stress with phase angle for node #N19.
```

B.5.9 STREPRR (node_name,S1,S2,S3,C1X,C1Y,C1Z,C2X,C2Y,C2Z, C3X,C3Y,C3Z,cosys);

```
PROPERTY streprr;
   S1,S2,S3             :  OPTIONAL real;
   C1X,C1Y,C1Z          :  OPTIONAL real;
   C2X,C2Y,C2Z          :  OPTIONAL real;
   C3X,C3Y,C3Z          :  OPTIONAL real;
   cosys                :  OPTIONAL coord;
OF
   node_name            :  SELECT(node,trn);
END_PROPERTY;
```

Outputs the principal stress (real) S1,S2,S3. The cosines C1X,C1Y,C1Z of the angles between S1 and the x-, y- and z- axes, respectively, are given. The cosines C2X, C2Y, C2Z of the angles between S2 and the x-, y- and z- axes, respectively, are given. The cosines C3X,C3Y,C3Z of the angles between S3 and the x-, y- and z-axes, respectively, are given.

Example:

```
STREPRR (#N20, 3.523E2, 1.263E2, 5.674E1, 0.23, 0.44, 0.11,
         0.56, 0.88, 0.92, 0.13, 0.53, 0.02,);
       outputs 3 principal stresses with 9 direction cosines for node #N20
       in global coordinates.
```

B.5.10 STREPRC (node_name,S1,S2,S3,PS1,PS2,PS3,C1X,C1Y,C1Z, C2X,C2Y,C2Z,C3X,C3Y,C3Z,cosys);

```
PROPERTY streprc;
   S1,S2,S3              :  OPTIONAL real;
   PS1,PS2,PS3           :  OPTIONAL real;
   C1X,C1Y,C1Z           :  OPTIONAL real;
   C2X,C2Y,C2Z           :  OPTIONAL real;
   C3X,C3Y,C3Z           :  OPTIONAL real;
   cosys                 :  OPTIONAL coord;
OF
   node_name             :  SELECT(node,trn);
END_PROPERTY;
```

Outputs the principal stress (complex) S1,S2,S3 as amplitudes and the corresponding phases PS1,PS2,PS3. The cosines C1X,C1Y,C1Z of the angles between S1 and the x-, y- and z-axes, respectively, are given. The cosines C2X,C2Y,C2Z of the angles between S2 and the x-, y- and z- axes, respectively, are given. The cosines C3X,C3Y,C3Z of the angles between S3 and the x-, y- and z-axes, respectively, are given.

Example:

```
STREPRC (#N20, 3.523E2, 1.263E2, 5.674E1, 179.4, 209.4, 198.9,
        0.23, 0.44, 0.11, 0.56, 0.88, 0.92, 0.13, 0.53, 0.02,);
       outputs 3 principal stresses with phase angles and 9 direction
       cosines for node #N20 in global coordinates.
```

B.5.11 STREENR (element_name,node_name,SX,SY,SZ, SXY,SXZ,SYZ,cosys);

```
PROPERTY streenr;
   node_name           :  SELECT(node,trn);
   SX,SY,SZ            :  OPTIONAL real;
   SXY,SXZ,SYZ         :  OPTIONAL real;
   cosys               :  OPTIONAL coord;
OF
   element_name        :  SELECT(element,tre);
END_PROPERTY;

TYPE
   element = SELECT (bar, beam2, tri3, tri6, tri7v,
                     quad4, quad8, quad9v, tetra4,
                     tetra10, tetra15v, penta6, penta15,
                     penta21v, hexa8, hexa20, hexa27v);
END_TYPE;
```

Outputs the 6 elemental related stress components SX,...,SYZ. They are calculated at the node named node_name in the element named element_name. The values are given in the coordinate system named cosys (default allowed). Omitted values are zero.

Example:

```
STREENR (#E123, #N20, 3.523E2, 1.263E2, 5.674E1, 3.231E1,
         1.446E1, 1.998E1, #col);
        outputs 6 stress components for element #E123 (location: node #N20)
        in coordinate system #col.
```

B.5.12 STREENC (element_name,node_name,SX,SY,SZ,SXY,SXZ,SYZ,PSX,PSY,PSZ, PSXY,PSXZ,PSYZ,cosys);

```
PROPERTY streenc;
   node_name           :  SELECT(node,trn);
   SX,SY,SZ            :  OPTIONAL real;
   SXY,SXZ,SYZ         :  OPTIONAL real;
   PSX,PSY,PSZ         :  OPTIONAL real;
   PSXY,PSXZ,PSYZ      :  OPTIONAL real;
   cosys               :  OPTIONAL coord;
OF
   element_name        :  SELECT(element,tre);
END_PROPERTY;

TYPE
   element = SELECT (bar, beam2, tri3, tri6, tri7v,
                     quad4, quad8, quad9v, tetra4,
                     tetra10, tetra15v, penta6, penta15,
                     penta21v, hexa8, hexa20, hexa27v);
END_TYPE;
```

Outputs the 6 elemental related stress components SX,...,SYZ as amplitudes and the corresponding phases PSX,...,PSYZ. They are calculated at the node named node_name in the element named element_name. The values are given in the coordinate system named cosys (default allowed). Omitted values are zero.

Example:

```
STREENC (#E123, #N20, 3.523E2, 1.263E2, 5.674E1, 3.231E1,
         1.446E1, 1.998E1, 0.012, -0.214, -0.038, -0.037,
         0.009, 0.155, #col);
```

outputs 6 complex stress components as amplitudes and phases (in radians) for element #E123 (location: node #N20) in coordinate system #col.

B.5.13 STREEIR (element_name,X,Y,Z,SX,SY,SZ,SXY,SXZ,SYZ,cosys);

```
PROPERTY streeir;
   X,Y,X                : real;
   SX,SY,SZ             : OPTIONAL real;
   SXY,SXZ,SYZ          : OPTIONAL real;
   cosys                : OPTIONAL coord;
OF
   element_name         : SELECT(element,tre);
END_PROPERTY;

TYPE
   element = SELECT (bar, beam2, tri3, tri6, tri7v,
                        quad4, quad8, quad9v, tetra4,
                        tetra10, tetra15v, penta6, penta15,
                        penta21v, hexa8, hexa20, hexa27v);
END_TYPE;
```

Outputs the 6 elemental related stress components SX,...,SYZ. They are calculated at the internal element point X,Y,Z in the element named element_name. The values are given in the coordinate system named cosys (default allowed). Omitted values are zero.

Example:

```
STREEIR (#E123, 1.0, 0.0, 2.5, 3.523E2, 1.263E2, 5.674E1,
         3.231E1, 1.446E1, 1.998E1, #col);
```
outputs 6 stress components for element #E123 (location given by coordinates 1,0,2.5) in coordinate system #col.

B.5.14 STREEIC (element_name,X,Y,Z,SX,SY,SZ,SXY,SXZ,SYZ, PSX,PSY,PSZ,PSXY,PSXZ,PSYZ,cosys);

```
PROPERTY streeic;
   X,Y,Z                 :  real;
   SX,SY,SZ              :  OPTIONAL real;
   SXY,SXZ,SYZ           :  OPTIONAL real;
   PSX,PSY,PSZ           :  OPTIONAL real;
   PSXY,PSXZ,PSYZ        :  OPTIONAL real;
   cosys                 :  OPTIONAL coord;
OF
   element_name          :  SELECT(element,tre);
END_PROPERTY;

TYPE
   element = SELECT (bar, beam2, tri3, tri6, tri7v, quad4,
                     quad8, quad9v, tetra4, tetra10,
                     tetra15v, penta6, penta15, penta21v,
                     hexa8, hexa20, hexa27v);
END_TYPE;
```

Outputs the 6 elemental related stress components SX,...,SYZ as amplitudes and the corresponding phases PSX,...,PSYZ. They are calculated at the internal element point X,Y,Z in the element named element_name. The values are given in the coordinate system named cosys (default allowed). Omitted values are zero.

Example:

```
STREEIC (#E123, 1.0, 0.0, 2.5, 3.523E2, 1.263E2, 5.674E1,
         3.231E1, 1.446E1, 1.998E1, 0.012, -0.214, -0.038,
         -0.037, 0.009, 0.155, );
```

outputs 6 complex stress components as amplitudes and phases (in radians) for element #E123 (location: coordinates 1,0,2.5) in global coordinate system.

B.5.15 STRANOR (node_name,EX,EY,EZ,EXY,EXZ,EYZ,cosys);

```
PROPERTY stranor;
   EX,EY,EZ             :  OPTIONAL real;
   EXY,EXZ,EYZ          :  OPTIONAL real;
   cosys                :  OPTIONAL coord;
OF
   node_name            :  SELECT(node,trn);
END_PROPERTY;
```

Outputs the 6 nodal strain components EX,...,EYZ. They are calculated at the node named node_name and given in the coordinate system named cosys (default allowed). Omitted values are zero.

Example:

```
STRANOR (#N17, 3.523E2, -1.263E2, 5.674E1, 6.599E1, 2.162E1,
         7.773E0, ,);
```

outputs 6 real strain components in global coordinates for node #N17.

B.5.16 STRANOC (node_name,EX,EY,EZ,EXY,EXZ,EYZ,PEX,PEY,PEZ, PEXY,PEXZ,PEYZ,cosys);

```
PROPERTY stranoc;
   EX,EY,EZ              :  OPTIONAL real;
   EXY,EXZ,EYZ           :  OPTIONAL real;
   PEX,PEY,PEZ           :  OPTIONAL real;
   PEXY,PEXZ,PEYZ        :  OPTIONAL real;
   cosys                 :  OPTIONAL coord;
OF
   node_name             :  SELECT(node,trn);
END_PROPERTY;
```

Outputs the 6 nodal strain components EX,...,EYZ as amplitudes and the corresponding phases PEX,...,PEYZ. They are calculated at the node named node_name and given in the coordinate system named cosys (default allowed). Omitted values are zero.

Example:

```
STRANOC (#N17, 3.523E2, -1.263E2, 5.674E1, 6.599E1, 2.162E1,
         7.773E0, 123.2, 97.5, 110.3, 99.5, 82.3, 101.1 ,);
```

outputs 6 complex strain components (phases in degrees) in global coordinates for node #N17.

B.5.17 STRAEQR (node_name,EE);

```
PROPERTY straeqr;
   EE                    :  real;
OF
   node_name             :  SELECT(node,trn);
END_PROPERTY;
```

Outputs the equivalent strain (real) EE for the node named node_name.

Example:

```
STRAEQR (#N19, 56.74);
       outputs equivalent strain for node #N19
```

B.5.18 STRAEQC (node_name,EE,PEE);

```
PROPERTY straeqc;
   EE                  :  real;
   PEE                 :  real;
OF
   node_name           :  SELECT(node,trn);
END_PROPERTY;
```

Outputs the equivalent strain (complex) EE as the amplitude and the corresponding phase PEE for the node named node_name.

Example:

```
STRAEQC (#N19, 56.74, 1.25);
       outputs equivalent strain with phase angle in radians for node #N19
```

B.5.19 STRAPRR (node_name,E1,E2,E3,C1X,C1Y,C1Z,C2X,C2Y,C2Z, C3X,C3Y,C3Z,cosys);

```
PROPERTY straprr;
   E1,E2,E3            :  OPTIONAL real;
   C1X,C1Y,C1Z         :  OPTIONAL real;
   C2X,C2Y,C2Z         :  OPTIONAL real;
   C3X,C3Y,C3Z         :  OPTIONAL real;
   cosys               :  OPTIONAL coord;
OF
   node_name           :  SELECT(node,trn);
END_PROPERTY;
```

Outputs the principal strain (real) E1,E2,E3. The cosines C1X,C1Y,C1Z of the angles between E1 and the x-, y- and z- axes, respectively, are given. The cosines C2X,C2Y,C2Z of the angles between E2 and the x-, y- and z-axes, respectively, are given. The cosines C3X,C3Y,C3Z of the angles between E3 and the x-, y- and z-axes, respectively, are given.

Example:

```
STRAPRR (#N20, 3.523E2, 1.263E2, 5.674E1, 0.23, 0.44, 0.11,
         0.56, 0.88, 0.92, 0.13, 0.53, 0.02,);
       outputs 3 principal strains with 9 direction cosines for node #N20
       in global coordinates.
```

B.5.20 STRAPRC (node_name,E1,E2,E3,PE1,PE2,PE3,C1X,C1Y,C1Z, C2X,C2Y,C2Z,C3X,C3Y,C3Z,cosys);

```
PROPERTY straprc;
   E1,E2,E3              :  OPTIONAL real;
   PE1,PE2,PE3           :  OPTIONAL real;
   C1X,C1Y,C1Z           :  OPTIONAL real;
   C2X,C2Y,C2Z           :  OPTIONAL real;
   C3X,C3Y,C3Z           :  OPTIONAL real;
   cosys                 :  OPTIONAL coord;
OF
   node_name             :  SELECT(node,trn);
END_PROPERTY;
```

Outputs the principal strain (complex) E1,E2,E3 as amplitudes and the corresponding phases PE1,PE2,PE3. The cosines C1X,C1Y,C1Z of the angles between E1 and the x-, y- and z-axes, respectively, are given. The cosines C2X,C2Y,C2Z of the angles between E2 and the x-, y- and z- axes, respectively, are given. The cosines C3X,C3Y,C3Z of the angles between E3 and the x-, y- and z-axes, respectively, are given.

Example:

```
STRAPRC (#N20, 3.523E2, 1.263E2, 5.674E1, 179.4, 209.4, 198.9, 0.23,
        0.44, 0.11, 0.56, 0.88, 0.92, 0.13, 0.53, 0.02,);
```
outputs 3 principal strains with phase angles and 9 direction cosines for node #N20 in global coordinates.

B.5.21 STRAENR (element_name,node_name,EX,EY,EZ, EXY,EXZ,EYZ,cosys);

```
PROPERTY straenr;
   node_name            :  SELECT(node,trn);
   EX,EY,EZ             :  OPTIONAL real;
   EXY,EXZ,EYZ          :  OPTIONAL real;
   cosys                :  OPTIONAL coord;
OF
   element_name         :  SELECT(element,tre);
END_PROPERTY;

TYPE
   element = SELECT (bar, beam2, tri3, tri6, tri7v, quad4,
                     quad8, quad9v, tetra4, tetra10,
                     tetra15v, penta6, penta15, penta21v,
                     hexa8, hexa20, hexa27v);
END_TYPE;
```

Outputs the 6 elemental related strain components EX,...,EYZ. They are calculated at the node named node_name in the element named element_name. The values are given in the coordinate system named cosys (default allowed). Omitted values are zero.

Example:

```
STRAENR (#E123, #N20, 3.523E2, 1.263E2, 5.674E1, 3.231E1,
         1.446E1, 1.998E1, #col);
        outputs 6 strain components for element #E123 (location: node #N20)
        in coordinate system #col.
```

B.5.22 STRAENC (element_name,node_name,EX,EY,EZ,EXY,EXZ,EYZ, PEX,PEY,PEZ,PEXY,PEXZ,PEYZ,cosys);

```
PROPERTY straenc;
   node_name            :  SELECT(node,trn);
   EX,EY,EZ             :  OPTIONAL real;
   EXY,EXZ,EYZ          :  OPTIONAL real;
   PEX,PEY,PEZ          :  OPTIONAL real;
   PEXY,PEXZ,PEYZ       :  OPTIONAL real;
   cosys                :  OPTIONAL coord;
OF
   element_name         :  SELECT (element,tre);
END_PROPERTY;

TYPE
   element = SELECT (bar, beam2, tri3, tri6, tri7v, quad4,
                     quad8, quad9v, tetra4, tetra10,
                     tetra15v, penta6, penta15, penta21v,
                     hexa8, hexa20, hexa27v);
END_TYPE;
```

Outputs the 6 elemental related strain components EX,...,EYZ as amplitudes and the corresponding phases PEX,...,PEYZ. They are calculated at the node named node_name in the element named element_name. The values are given in the coordinate system named cosys (default allowed). Omitted values are zero.

Example:

```
STRAENC (#E123, #N20, 3.523E2, 1.263E2, 5.674E1, 3.231E1, 1.446E1,
      1.998E1, 0.012, -0.214, -0.038, -0.037, 0.009, 0.155, #col);
```

outputs 6 complex strain components as amplitudes and phases (in radians) for element #E123 (location: node #N20) in coordinate system #col.

B.5.23 STRAEIR (element_name,X,Y,Z,EX,EY,EZ,EXY,EXZ,EYZ,cosys);

```
PROPERTY straeir;
   X,Y,Z                  :  real;
   EX,EY,EZ               :  OPTIONAL real;
   EXY,EXZ,EYZ            :  OPTIONAL real;
   cosys                  :  OPTIONAL coord;
OF
   element_name           :  SELECT (element,tre);
END_PROPERTY;

TYPE
   element = SELECT (bar, beam2, tri3, tri6, tri7v, quad4,
                     quad8, quad9v, tetra4, tetra10,
                     tetra15v, penta6, penta15, penta21v,
                     hexa8, hexa20, hexa27v);
END_TYPE;
```

Outputs the 6 elemental related strain components EX,...,EYZ. They are calculated at the internal element point X,Y,Z in the element named element_name. The values are given in the coordinate system named cosys (default allowed). Omitted values are zero.

```
STRAENC (#E123, #N20, 3.523E2, 1.263E2, 5.674E1, 3.231E1, 1.446E1,
          1.998E1, 0.012, -0.214, -0.038, -0.037, 0.009, 0.155, #col);
```

outputs 6 complex strain components as amplitudes and phases (in radians) for element #E123 (location: node #N20) in coordinate system #col.

B.5.24 STRAEIC (element_name,X,Y,Z,EX,EY,EZ,EXY,EXZ,EYZ, PEX,PEY,PEZ,PEXY,PEXZ,PEYZ,cosys);

```
PROPERTY straeic;
   X,Y,Z              :  real;
   EX,EY,EZ           :  OPTIONAL real;
   EXY,EXZ,EYZ        :  OPTIONAL real;
   PEX,PEY,PEZ        :  OPTIONAL real;
   PEXY,PEXZ,PEYZ     :  OPTIONAL real;
   cosys              :  OPTIONAL coord;
OF
   element_name       :  SELECT (element,tre);
END_PROPERTY;

TYPE
   element = SELECT (bar, beam2, tri3, tri6, tri7v, quad4,
                     quad8, quad9v, tetra4, tetra10,
                     tetra15v, penta6, penta15, penta21v,
                     hexa8, hexa20, hexa27v);
END_TYPE;
```

Outputs the 6 elemental related strain components EX,...,EYZ as amplitudes and the corresponding phases PEX,...,PEYZ. They are calculated at the internal element point X,Y,Z in the element named element_name. The values are given in the coordinate system named cosys (default allowed). Omitted values are zero.

Example:

```
STRAEIC (#E123, 1.0, 0.0, 2.5, 3.523E2, 1.263E2, 5.674E1,
         3.231E1, 1.446E1, 1.998E1, 0.012, -0.214, -0.038,
         -0.037, 0.009, 0.155, );
```

outputs 6 complex strain components as amplitudes and phases (in radians) for element #E123 (location: coordinates 1,0,2.5) in global coordinate system.

B.5.25 NODFORCER(node_name,ITYP,RX,RY,RZ,MX,MY,MZ,cosys);

```
PROPERTY nodforcer;
   ITYP                 :  integer;
   RX,RY,RZ             :  OPTIONAL real;
   MX,MY,MZ             :  OPTIONAL real;
   cosys                :  OPTIONAL coord;
OF
   node_name            :  SELECT (node,trn);
END_PROPERTY;
```

Outputs the 3 force components RX,RY,RZ and the 3 moment components MX,MY,MZ calculated at the node named node_name. They are given in the coordinate system cosys. The type of the force is given by the parameter ITYP where;

```
        ITYP = 1  indicates a spring force
             = 2  indicates a damping force
             = 3  indicates an inertia force.
```

Example:

```
NODFORCER (#N523, 2, 3.523E2, 1.263E2, 5.674E1, 3.231E1,
           1.446E1, 1.998E1, #col);
```

outputs 6 damping force components for node #E523 in coordinate system #col.

B.5.26 NODFORCEC (node_name,ITYP,RX,RY,RZ,MX,MY,MZ,PRX,PRY,PRZ, PMX,PMY,PMZ,cosys);

```
PROPERTY nodforcec;
   ITYP                :  integer;
   RX,RY,RZ            :  OPTIONAL real;
   MX,MY,MZ            :  OPTIONAL real;
   PRX,PRY,PRZ         :  OPTIONAL real;
   PMX,PMY,PMZ         :  OPTIONAL real;
   cosys               :  OPTIONAL coord;
OF
   node_name           :  SELECT (node,trn);
END_PROPERTY;
```

Outputs the 3 force components RX,RY,RZ and the 3 moment components MX,MY,MZ as amplitudes and the corresponding phases PRX,...,PMZ calculated at the node named node_name. They are given in the coordinate system cosys. The type of the force is given by the parameter ITYP where

```
        ITYP = 1  indicates a spring force
             = 2  indicates a damping force
             = 3  indicates an inertia force.
```

Example:

```
NODFORCEC (#N523, 2, 3.523E2, 1.263E2, 5.674E1, 3.231E1,
          1.446E1, 1.998E1, 123.2, 97.5, 110.3, 99.5,
          82.3, 101.1, #col);
```

outputs 6 complex damping force components with phase angles in degrees for node #E523 in coordinate system #col.

B.5.27 EIGFR ((LIST_OF_EIGENFREQ));

```
PROPERTY eigfr;
   LIST_OF_EIGENFREQ    :  SET OF real;
END_PROPERTY;
```

Outputs a list of calculated eigenfrequencies.

Example:

```
EIGFR ((11.224, 17.995, 25.381, 25.386, 49.921, 60.801));
       outputs the first 6 eigenfrequencies as a list.
```

B.5.28 DAMPRAT ((LIST_OF_DAMPRAT));

```
PROPERTY nodflux;
   LIST_OF_DAMPRAT      :  SET OF real;
END_PROPERTY;
```

Outputs a list of calculated damping ratios, i.e. parts of the critical damping.

```
DAMPRAT ((0.0224, 0.0995, 0.0381, 0.0386, 0.0921, 0.0801));
       outputs the damping ratios of 6 eigenfrequencies as a list.
```

B.5.39 NODTEMP (node_name,TE);

```
PROPERTY nodtemp;
   TE                   :  real;
OF
   node_name            :  SELECT (node,trn);
END_PROPERTY;
```

Outputs the calculated temperature TE at the node named node_name.

Example:

```
NODTEMP (#N99, 82.35);
       outputs the temperature for node #N99.
```

B.5.30 NODFLUX (element_name,(list_of_nodes),HF);

```
PROPERTY nodflux;
   list_of_nodes        : LIST [1 TO 4] OF gnode;
   HF                   :  real;
OF
   element_name         :  SELECT (element,tre);
END_PROPERTY;

TYPE
   element = SELECT (bar, beam2, tri3, tri6, tri7v, quad4,
                        quad8, quad9v, tetra4, tetra10,
                        tetra15v, penta6, penta15, penta21v,
                        hexa8, hexa20, hexa27v);
END_TYPE;

TYPE
   gnode = select (node, trn);
END_TYPE;
```

Outputs the calculated heat flux HF over the defined element surface given by up to four node names, in list_of_nodes, in the element named element_name.

Example:

```
NODFLUX (#E77, (#N9,#N10,#N11,#N12), 9.3335E-03);
       outputs heat flux through one surface of element #E77 described by
       nodes #N9,#N10,#N11,#N12.
```

B.5.31 RC (rcname:ICR,(LIST_OF_ROWCOL_PAIR), (LIST_OF_NUM_PAIR));

```
ENTITY rc;
   ICR                  :  integer;
   LIST_OF_ROWCOL_PAIR  :  SET OF integer;
   LIST_OF_NUM_PAIR     :  SET OF integer;
END_ENTITY;
```

Defines one row/column specification which is used in the MATSHP statement. ICR is the row/column number for which the non-zero element positions are given. The two list parameters LIST_OF_ROWCOL_PAIR and LIST_OR_NUM_PAIR define the non-zero positions. The first list indicates the starting column/row position of the sequences of non-zero elements in this row/column. The second list indicates the lengths of the sequences.

It is recommended to use RC as an embedded entity.

Example:

```
MATSHP (#Form5: 8,8, ROWW,COMP,FULL, (RC(1,(1,5),(1,1)), RC(2,(2,6),(1,1)),
        RC(3,(3,7),(1,1)), RC(4,(1,6,8),(4,1,1)),
        RC(6,(4),(1)), RC(8,(2),(5)) )  );
```

defines the form of a complex 8x8 sparse matrix with the following appearance:

```
x . . . x . . .
. x . . . x . .
. . x . . . x .
x x x x . x . x
. . . . . . . .
. . . x .  .. .
. . . . . . . .
. x x x x x . .
```

B.5.32 MATSEQ (name:n/e_list_ref);

```
ENTITY matseq;
   n/e_list_ref         :  SELECT (nodelist, elemlist);
END_ENTITY;
```

Defines the relationship between internal and external node or element numbering when matrices have been stored in an order to minimise solution time (in an FE analysis program).

The referred NODELIST or ELEMLIST statement may appear in the result section of the neutral file.

Example:

```
MATSEQ (#seq1: NODELIST ((#1,#2,#3,#5,#6,#7,#8,#9,#4,#10,#11,#12,
               #13,#14,#15,#16,#17,#18,#19,#20)) );
```

defines a sequence of nodes (to be used in a matrix definition) with node #4 moved behind node #9.

B.5.33 MATBAND (form:NR,NC,ROWCOL_TYPE,RECO_TYPE,STORE_TYPE,IBWLOW,IBWUP);

```
ENTITY matband;
   NR                    :  integer;
   NC                    :  integer;
   ROWCOL_TYPE           :  ENUMERATION OF (ROWW,COLW);
   RECO_TYPE             :  ENUMERATION OF (REAL,COMP);
   STORE_TYPE            :  ENUMERATION OF (FULL,SYMM,LOTR,UPTR,DIAG);
   IBWLOW                :  integer;
   IBWUP                 :  integer;
END_ENTITY;
```

Defines the form of a band matrix with NR rows and NC supplied. The next three arguments are subkeywords. The first type indicates row- or columnwise storage of the matrix. The second type indicates a real or a complex matrix. The third type indicates the storage mode (or the type of the matrix), where

FULL a full matrix
SYMM a symmetric matrix but all the elements are given
LOTR a lower triangle matrix, i.e. symmetric
UPTR an upper triangle matrix, i.e. symmetric
DIAG a diagonal matrix

IBWLOW and IBWUP denote the lower and upper bandwidth (including the main diagonal).

Example:

MATBAND (#Form3: 389,389, ROWW,REAL,SYMM,10,10);
defines the form of a real band matrix of size 389. For each row 19 values are stored (9 values left of the main diagonal, the main diagonal, 9 values right of the main diagonal).

B.5.34 MATGEN (form:NR,NC,ROWCOL_TYPE,RECO_TYPE,STORE_TYPE);

```
ENTITY matgen;
   NR                     :  integer;
   NC                     :  integer;
   ROWCOL_TYPE            :  ENUMERATION OF (ROWW,COLW);
   RECO_TYPE              :  ENUMERATION OF (REAL,COMP);
   STORE_TYPE             :  ENUMERATION OF (FULL,SYMM,LOTR,UPTR,DIAG);
END_ENTITY;
```

Defines the form of a general matrix with NR rows and NC columns, i.e. all the components of the matrix have to be supplied. The last three arguments are subkeywords. The first type indicates row- or columnwise storage of the matrix. The second type indicates a real or a complex matrix. The third type indicates the storage mode (or the type of the matrix), where

FULL	a full matrix
SYMM	a symmetric matrix but all the elements are given
LOTR	a lower triangle matrix, i.e. symmetric
UPTR	an upper triangle matrix, i.e. symmetric
DIAG	a diagonal matrix

Example:

MATGEN (#Form2: 6,6, COLW,REAL,SYMM);

defines the form of a real matrix with 6 rows and columns where only the upper triangle is given columnwise. The following scheme shows the distribution of the 21 input values in the matrix:

```
 1  2  4  7 11 16
 2  3  5  8 12 17
 4  5  6  9 13 18
 7  8  9 10 14 19
11 12 13 14 15 20
16 17 18 19 20 21
```

B.5.35 MATSHP (form:NR,NC,ROWCOL_TYPE,RECO_TYPE,STORE_TYPE, (list_of_rowcol_spec));

```
ENTITY matshp;
   NR                    :  integer;
   NC                    :  integer;
   ROWCOL_TYPE           :  ENUMERATION OF (ROWW,COLW);
   RECO_TYPE             :  ENUMERATION OF (REAL,COMP);
   STORE_TYPE            :  ENUMERATION OF (FULL,SYMM);
   list_of_rowcol_spec   :  SET OF rc;
END_ENTITY;
```

Defines the form of a sparse matrix with NR rows and NC columns. The next three arguments are types. The first type indicates row- or columnwise storage of the matrix. The second type indicates a real or a complex matrix. The third type indicates the storage mode (or the type of the matrix), where

FULL a full matrix
SYMM a symmetric matrix but all the elements are given

The parameter list_of_rowcol_spec specifies the non-zero matrix element positions. The specification is given separately for each row/column by the RC statement.

Example:

```
MATSHP (#Form4: 19,19, COLW,REAL,UPTR, (#rc1, #rc2, #rc3, #rc4,
        #rc5, #rc6, #rc7, #rc8, #rc9) );
```

defines the form of a sparse matrix of size 19. The upper triangle is given columnwise. There are 9 locations with non-zero elements. These are described by separate RC statements.

B.5.36 MATDEF (name:PURPOSE,shape,'TEXT',PARTITION,seq,e/o_ref);

```
ENTITY matdef;
   PURPOSE              :  ENUMERATION OF (XSTIFF, XDAMP, XMASS, XGEN);
   shape                :  SELECT (matshp,matband,matgen);
   TEXT                 :  OPTIONAL string;
   PARTITION            :  OPTIONAL ENUMERATION OF (PARTITION);
   seq                  :  OPTIONAL matseq
   e/o_ref              :  SELECT (element, object);
END_ENTITY;

TYPE
   object = SELECT (collect, assem, scond,copy);
   element = SELECT (bar, beam2, tri3, tri6, tri7v, quad4,
                     quad8, quad9v, tetra4, tetra10,
                     tetra15v, penta6, penta15, penta21v,
                     hexa8, hexa20, hexa27v);
END_TYPE;
```

A matrix definition keyword. Gives a name by which a matrix may be referenced, and associates it with an element or object.

PURPOSE	XSTIFF a stiffness matrix XDAMP a damping matrix XMASS a mass matrix XGEN a general matrix
shape	a reference to a previously defined form
'TEXT'	an optional text description
PARTITION	if this parameter is omitted, then the matrix is not partitioned.
seq	a reference to a MATSEQ statement
e/o_ref	a reference to the element or object this matrix should be applied to

Example:

MATDEF (#[K]: XSTIFF, #Form2, 'Stiffness Matrix of part # 2', , ,#OB2);
defines for the structure #OB2 the stiffness matrix #[K] with its form given by #Form2. No partitioning was applied and the sequence defaults to the natural sequence (1,2,3,...).

B.5.37 MATVAL (name,form,(LIST_OR_REAL_VALUES), (LIST_OF_IMAG_VALUES));

```
PROPERTY matval;
   form                 :  SELECT (matgen, matband, matshp);
   LIST_OF_REAL_VALUES  :  LIST OF real;
   LIST_OF_IMAG_VALUES  :  OPTIONAL LIST OF real;
OF
   name                 :  matdef
END_ENTITY;
```

Assigns to a defined matrix the values of the components. If the matrix is real the list of imaginary values is omitted.

Example:

```
MATVAL (#[K], #Form2, (3.456789E8, -2.876543E8,3.918473E8,0.,
        3.216392E7, 6.649921E8, 0.,0.,1.875391E8,5.251892E8,
        9.432198E7,0.,0., 3.543213E7,9.765246E8) ,);
       outputs 15 values of the 5x5 stiffness matrix #[K].
```

B.5.38 CLOSEFERES (set_name);

```
PROPERTY closeferes;
OF
   set_name             : OPENFERES;
END_PROPERTY;
```

Closes this result block.

B.6 Notes

Note (1)
Unless stated otherwise, wherever a reference is made to a coordinate system (COORD keyword) this may be omitted, indicating that all coordinates are in the global cartesian coordinate system.

Note (2)
A load of dimension n cannot be attached to an element of dimension n-1, but it can be attached to an element of dimension n+1 (or n+2) if edges or surfaces of the element are specified.

Note (3)
Dynamic loads are defined as point loads with one component. A load which acts in a direction not parallel to a coordinate axis must be given as two (or three) dynamic loads.

Note (4)
The term 'thermal analysis' refers to that type of analysis where temperatures and heat flows can be considered the output of that analysis. The term 'static analysis' refers to that type of analysis where stresses and strains can be considered the output of the analysis.

Note (5)
All output requests imply a request for computation. It is not possible at present to request computation but no output.

B.7 Keyword Cross Reference

Keyword	Section
ALLELEMS	B.4.13
ALLNODES	B.4.12
ANCASE	B.5.2
ANCEIGV	B.4.59
ANFRESP	B.4.60
ANISO2D	B.3.8
ANISO3D	B.3.9
ANREIGV	B.4.58
ANSTATIC	B.4.57
ANSTEMP	B.4.62
ANTRESP	B.4.61
ASSEM	B.4.9
BAR	B.3.31
BEAM2	B.3.32
BEAMG	B.3.12
BEAMI	B.3.13
BEAML	B.3.14
BEAMO	B.3.17
BEAMR	B.3.18
BEAMT	B.3.16
BEAMU	B.3.15
CDOFLIST	B.4.3
CLOSEANA	B.4.74
CLOSEFERES	B.5.38
CLOSEMODEL	B.3.48
COLLECT	B.4.2
CONVEC	B.4.49
COORD	3.3
COPY	B.4.8
COUPLELIST	B.4.6
DAMP	B.3.20
DAMPER	B.3.29
DAMPRAT	B.5.28
DCOND	B.4.5
DISCRLOAD	B.4.36
DISPC	B.5.4
DISPR	B.5.3
DYNLOAD	B.4.35
EDGELOAD	B.4.40
EIGFR	B.5.27
ELEMENTAL	B.4.17
ELEMLIST	B.4.15
ELEMLOAD	B.4.46
FIRSTFR	B.4.67
FLUX	B.4.50
FREEDOM	B.3.4
FREQOUT	B.4.69
FREQRANGE	B.4.53
FREQRG	B.4.24
GRAVLOAD	B.4.47
HEXA20	B.3.46
HEXA27V	B.3.47
HEXA8	B.3.45
IMPULS	B.4.26

ISO....................B.3.6

ISOFULL................B.3.7

LOADL1.................B.4.38

LOADL2.................B.4.39

LOADP1.................B.4.22

LOADP2.................B.4.23

LOADS1.................B.4.41

LOADS2.................B.4.42

LOADV1.................B.4.44

LOADV2.................B.4.45

MASS...................B.3.28

MATBAND................B.5.33

MATDEF.................B.5.36

MATERIAL...............B.3.10

MATGEN.................B.5.34

MATSEQ.................B.5.32

MATSHP.................B.5.35

MATVAL.................B.5.37

MDCONST................B.4.55

MODALDAMP..............B.4.56

MODELIST...............B.4.68

MPC....................B.4.20

NOCOUPLE...............B.4.7

NODAL..................B.4.16

NODE...................B.3.5

NODELIST...............B.4.14

NODELOAD...............B.4.37

NODFLUX................B.5.30

NODFORCEC..............B.5.26

NODFORCER..............B.5.25

NODTEMP................B.5.29

OPENANA................B.4.1

OPENFERES..............B.5.1

OPENMODEL..............B.3.1

OR1....................B.3.25

OR3....................B.3.26

ORIENT.................B.3.27

OUTDISP................B.4.63

OUTEIGFR...............B.4.70

OUTFLUX................B.4.73

OUTFORCE...............B.4.66

OUTMODESH..............B.4.71

OUTSTRAIN..............B.4.65

OUTSTRESS..............B.4.64

OUTTEMP................B.4.72

PBAR...................B.3.11

PENTA15................B.3.43

PENTA21V...............B.3.44

PENTA6.................B.3.42

PMASS..................B.3.21

PRESET1................B.4.18

PRESET2................B.4.19

PROPERTY...............B.3.24

QUAD4..................B.3.36

QUAD8..................B.3.37

QUAD9V.................B.3.38

QVOL...................B.4.51

QVOLP..................B.4.52

RANDOM..................B.4.34

RC......................B.5.31

RECTAN..................B.4.27

SAWTOO..................B.4.28

SCOND...................B.4.4

SINUS...................B.4.29

SPRING..................B.3.30

STARTE..................B.4.33

STARTL..................B.4.32

STEADY..................B.4.25

STIFF...................B.3.19

STRAEIC.................B.5.24

STRAEIR.................B.5.23

STRAENC.................B.5.22

STRAENR.................B.5.21

STRAEQC.................B.5.18

STRAEQR.................B.5.17

STRANOC.................B.5.16

STRANOR.................B.5.15

STRAPRC.................B.5.20

STRAPRR.................B.5.19

STREEIC.................B.5.14

STREEIR.................B.5.13

STREENC.................B.5.12

STREENR.................B.5.11

STREEQC.................B.5.8

STREEQR.................B.5.7

STRENOC.................B.5.6

STRENOR.................B.5.5

STREPRC B.5.10

STREPRR B.5.9

SURFLOAD B.4.43

SWEEPE B.4.31

SWEEPL B.4.30

TEMP B.4.21

TEMPLOAD B.4.48

TETRA10 .... B.3.40

TETRA15V B.3.41

TETRA4 .... B.3.39

THICK B.3.22

THICKV B.3.23

TIMERANGE B.4.54

TRE B.4.11

TRI3 .... B.3.33

TRI6 .... B.3.34

TRI7V B.3.35

TRN B.4.10

UNITS B.4.2

C Implementation

C.1 Processors for FE data

Several interfaces are needed for storage and transfer of FE data. The origin of all data to be used in the FE world is in the design process. So we must base our first interface on product definition data coming out of CAD systems. These data are available on CAD*I neutral files of CAD type and have to be transferred to a FEM preprocessor.

The FEM preprocessor generates meshes, completes them with additional data and allows manipulation and checks. At last it formats the data for one of the various FEM analysis programs. Here we introduce our next interface which allows to store FE input data on a neutral file. The respective interface processors belong either to one of the FE preprocessors or to one of the FE analysis programs. The former writes a CAD*I neutral file of FEM type, the latter reads it.

The analysis program produces an output file which can again be converted to a neutral file. This is our next interface. Its preprocessors belong either to one FE analysis program or one FE postprocessor. The neutral file for FE output data may also contain FE input data. Also possible is to extend an FE input neutral file with FE output data.

The three interfaces described here are shown in fig. 12.

The neutral files are represented by data base symbols, the interface processors by arrows and the FE programs by boxes.

The concept of neutral files helps to minimise the number of necessary interface processors. Instead of combining each package with each other we need only combine each package to a neutral file processor, see fig. 13.

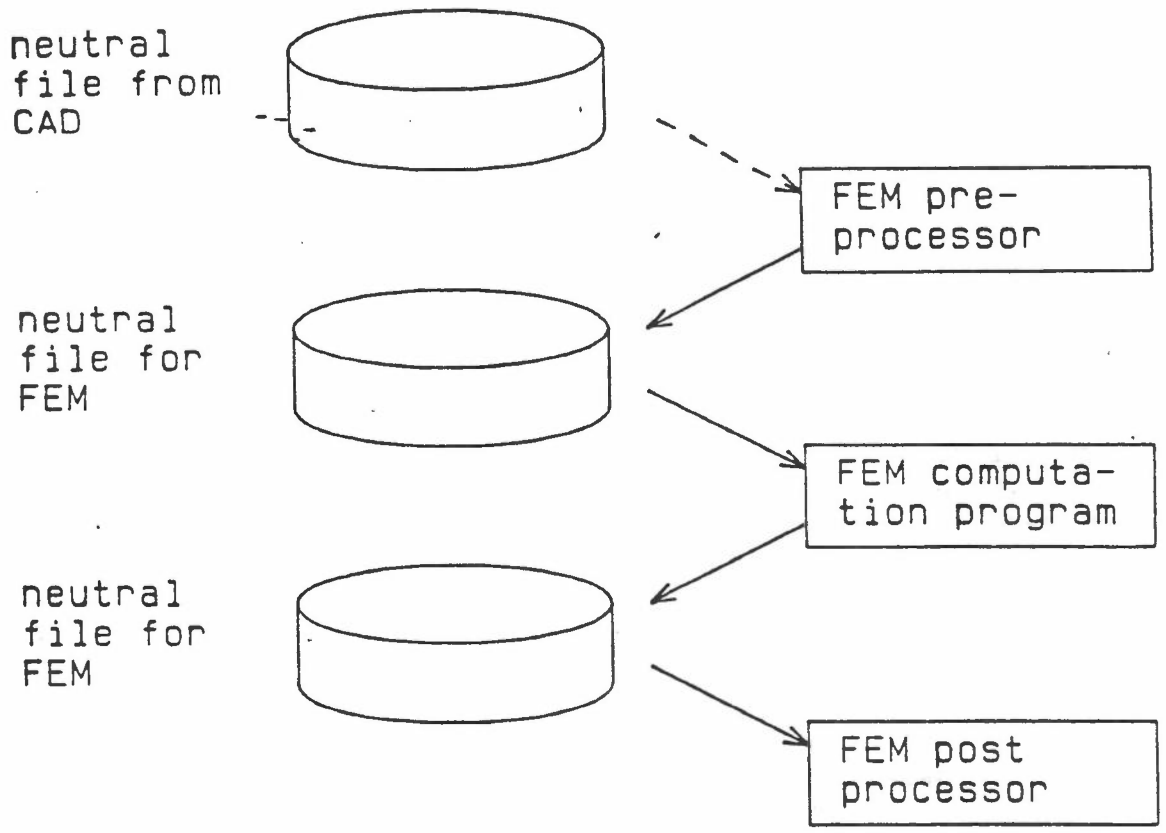


Figure 12. Interfaces to FE Processors.

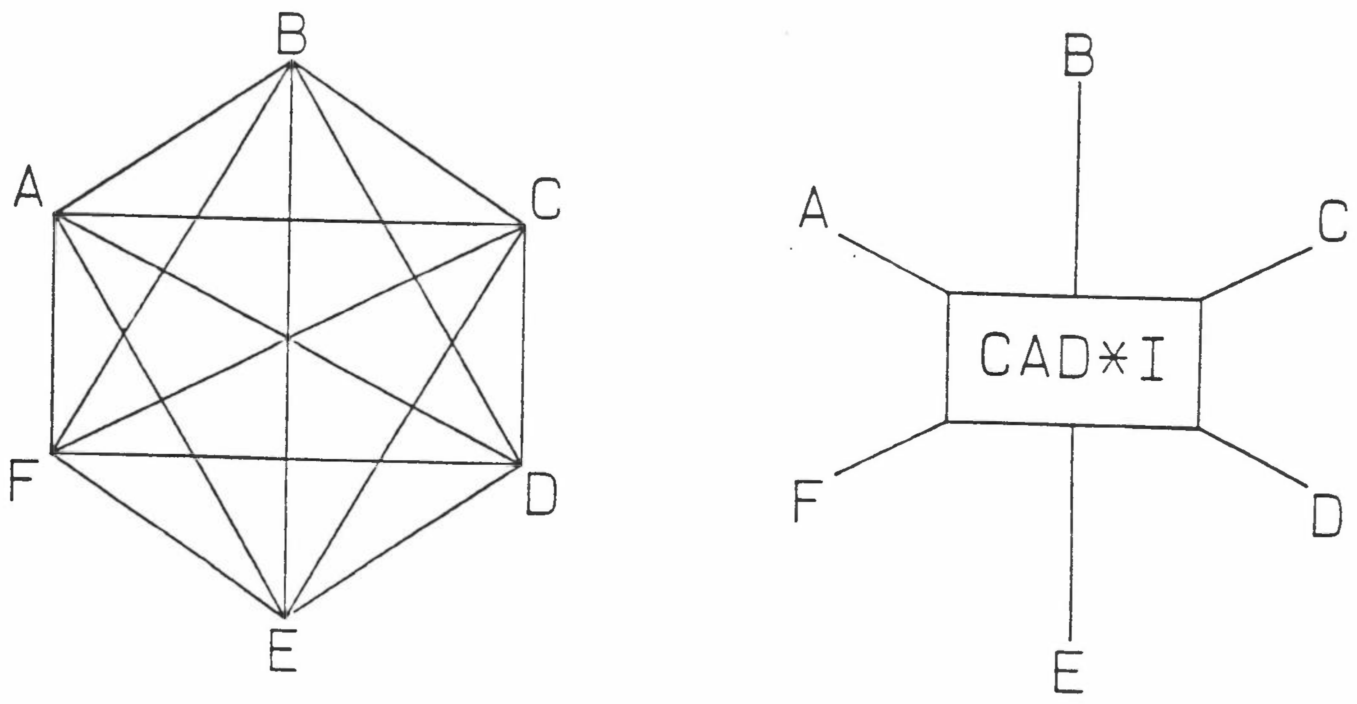


Figure 13. Reducing the Number of Interfaces.

C.2 Software Developed by the Partners

In order to prevent CAD*I from being a theoretical project, tasks which require realization and tests are included in the work-programme. This means that WG6 develops programs that read, write and handle the neutral files. These programs are described in this section and illustrated graphically in figure 14.

C.2.1 Software developed by NEH

C.2.1.1 Program File Handler

Purpose: To read a CAD*I metafile which contains multiple neutral files eg. letters, CAD and FEM neutral files: These files are identified and split into separated files. On demand from the user they may be translated into either PIGS or PAFEC input files.

The program detects via the file header the type of the neutral file. When a letter file is detected the user is asked if he wants to see the letter. If yes, the letter is shown on the screen. The letters are stored in a list file.

When a WG2 CAD file is detected the user is asked if he wants to translate the file into PIGS input. If yes, the program NEH61 is called to perform the task. When a WG6 FEM file is detected the user is asked if he wants to translate the file into PAFEC input. If yes, the program NEH63 is called to perform the task.

Language: Fortran 77
Operating system: Apollo Computer Aegis rel. 9.7

Size:	object code	21 Kb
	source code	36 Kb

Machine dependencies: Unit numbers for terminal I/O
open statements

Status (per 1.8.1988): The program is ready for use. The program uses the programs NEH61 and NEH63 (and therefore also RDNF and NF2FG from RAL).

Input: CAD*I metafile
Output: letter files, PIGS journal files (ie. PIGS commands) and PAFEC input files.

Extendability: -

Ownership: NEH Consulting Enginners ApS, Denmark
Author : Jes Mou Jessen, NEH Consulting Engineers

C.2.1.2 Program NEH61

Purpose: To read and interpret a CAD*I WG2 neutral file (version 3.3) and produce a corresponding PIGS level 4.1 (Pafec FE preprocessor) input file. I.e. a file which contains PIGS commands. The program translates the WG2 CAD information into appropriate PIGS commands.

Language: Fortran 77
Operating system: Apollo computer Aegis rel 9.7
Size: object code 30 Kb
source code 35 Kb
Machine dependencies: Unit numbers for terminal I/O
open statements

Status (per 1.8.1988): The program is ready for use.
Currently, the program covers some WG2 primitives such as
BOX,
SOLID_SPHERE,
SOLID_CYLINDER,
TRUNCATED_CONE,
TRUNCATED_PYRAMID
REGULAR_PRISM,
SOLID_TORUS

Input: CAD*I neutral file version WG2 3.3
Output: PIGS input file, ie. the xxxx.jnl file
They are translated into FE nodes, replicas, and FE elements.
Extendability: -

Ownership: NEH Consulting Engineers ApS, Denmark
Author : Jes Mou Jessen, NEH Consulting Engineers

The program uses RDNF and NF2FG which are programs written by RAL.

C.2.1.3 Program NEH62

Purpose: To access the database of the FEA system PAFEC (Pafec Ltd., UK) and extract the relevant information. Then a neutral file is produced.

The program is divided into three parts. Part No.1 handles the model definition, part No.2 handles the analysis definition and part No.3 handles the computed results. The program generates automatically the necessary references between these parts.

Language: Fortran 77
Operating system: Apollo Computer Aegis rel. 9.7
Size: Object code 101 Kb
Source code 340 Kb
Machine dependencies: Unit numbers for terminal I/O
open statements

Status (per 1.8.1988): The program is ready for use.
Currently, program part No.1 covers most of the functionality in the keyword definition for model description. Program part No.2 covers the

static load cases, different analysis requests, output requests. Program part No.3 covers the displacements, mode shapes and eigenfrequencies.

Input: PAFEC Internal database, level 6.2, ie. the xxxx.bs file Output: WG6 CAD*I neutral file with FE information Extendability: The program is able to interface both the pre-processor program PIGS and the analysis program, because they share a common database.

Ownership: NEH Consulting Engineers ApS, Denmark
Author : Jes Mou Jessen, NEH Consulting Engineers

C.2.1.4 Program NEH63

Purpose: To generate a PAFEC input data file, level 6.2 from the CAD*I WG6 neutral file. The program handles the keywords concerning the model and analysis definitions.

```
Language: Fortran 77
Operating system: Apollo Computer Aegis rel. 9.7
Size:     Object code   105 Kb
          Source code   126 Kb
Machine dependencies: Unit numbers for terminal I/O
                      open statements
```

Status (per 1.8.1988): The program is ready for use.
Currently, the program covers most of the functionality in the keyword definition for model description. The static loads are covered and some of the analysis requests.

Input: WG6 neutral file with FE information
Output: PAFEC input data, ie. xxxx.dat file
Extendability: -

Ownership: NEH Consulting Engineers ApS, Denmark
Author : Troels Ladefoged, NEH Consulting Engineers

In the program is used the parser and scanner program RDNF which is written by RAL.

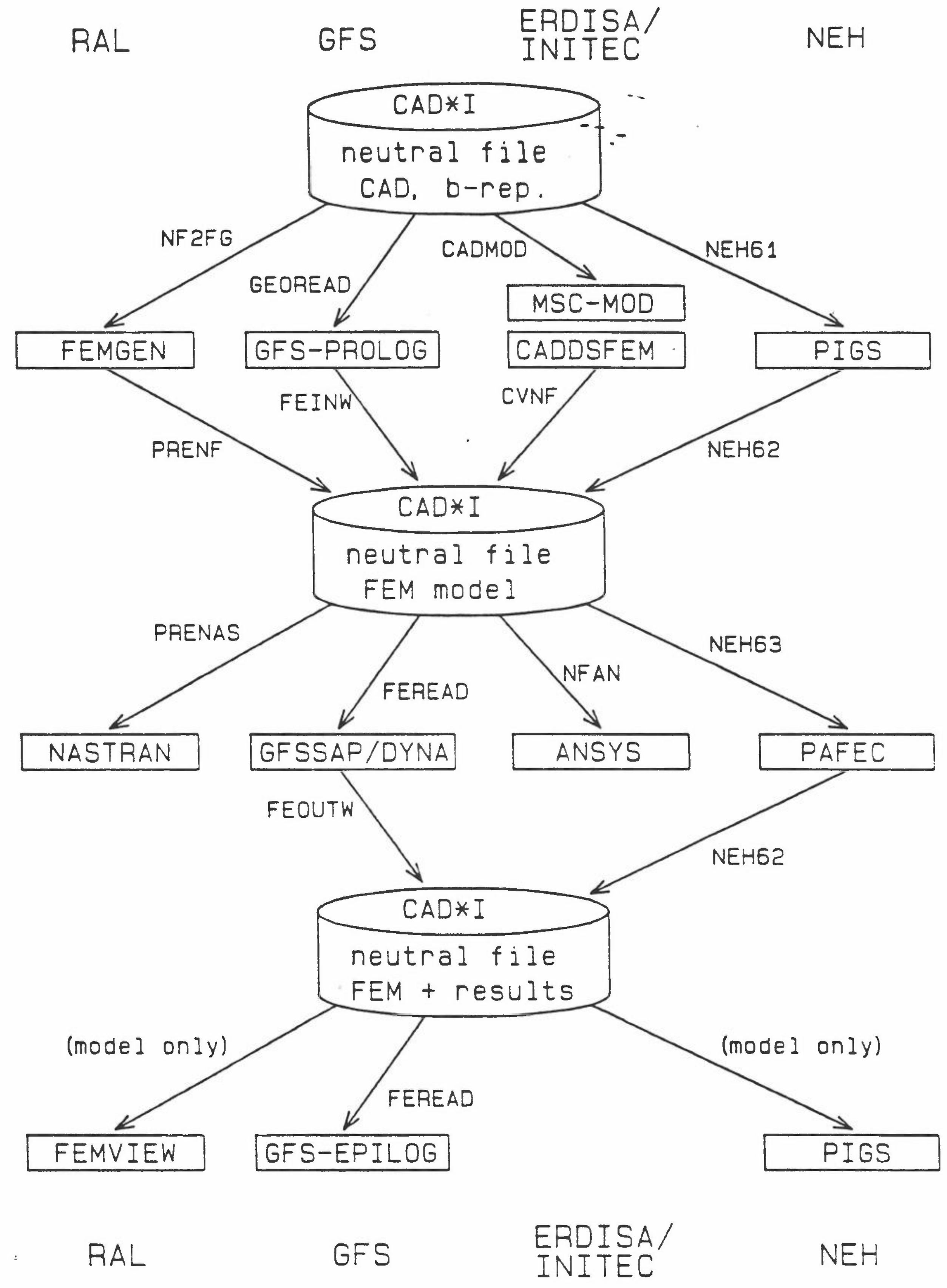


Figure 14. Software developed by the partners.

C.2.2 Software developed by RAL

C.2.2.1 Program RDNF

Purpose: Program RDNF can be used to read the neutral file as defined in the CAD*I project for exchanging information between CAD or finite element systems. RDNF reads the neutral file, checks for syntactical and semantical errors and stores the information from the neutral file in an easily accessible data structure. RDNF uses an LR(1) parser and a lexical analyser for syntax checking and organising the reading of the file. The syntax of the neutral file is described in Backus-Naur Form notation in section B.1.3. These BNF productions are read by the parser generator program and transformed into a set of tables used by RDNF. While reading the information from the neutral file all information is checked for correctness as far as possible and then stored in a database. To facilitate the use of the database special access routines are provided with the program.

The semantical information on keywords and their parameters is maintained in the keyword description file which is separate from program RDNF. This file is read at the beginning of the run of RDNF and its information is internally stored. This has the advantage that the program is independent from any particular application area as each application can have a keyword description file of its own. Also, if the description of a single keyword is changed it is not necessary to modify the program RDNF as all relevant information is stored outside the program.

If the physical file has been read successfully, control at the end of the run is passed to a subroutine which is to be provided by the user. It is envisaged that this subroutine will interrogate the database and pass the data from the database on to the applications program.

Status: The program is currently under test by the partners in the CAD*I project.

Size: source 80k

object, depends on the size of the model

Machine Dependencies: use of certain file units for terminal I/O.

Author: Jan Van Maanen, RAL.

Implementation: Standard Fortran77

C.2.2.2 PRENAS

Purpose: To read a CAD*I neutral file containing model and analysis data and to write out a complete Nastran input deck. It uses RDNF to read the neutral file and store the data, it then accesses this data, performs various transformations and writes out the data in a form suitable as input to Nastran.

Status: This program is currently under development. Already implemented are executive and case control for simple static analyses, coordinate systems, grid cards, elements, material and physical properties, single point constraints, preset displacements and point loads. Work is in progress on surface and body loads.

Implementation: Fortran-77 on PRIME.

Author: Debbie Thomas, RAL.

C.2.2.3 PRE-NF

Name: PRE-NF.FTN77

Purpose:
To write a neutral file from information in a FEMGEN (or FAMbuild) database.
Author: Michael Mead, Rutherford Appleton Laboratory.
Size: 178 kbytes
Machine Dependencies:
1. Use of escape code in subroutine BELL;
2. Offset of 176 used when converting integers to character and back. This may easily be changed by altering the assignment of variable ICOFF in the main program.
Development Status:
Ready to be used.
Language: Fortran-77

C.2.2.4 NF2FG

Name: NF2FG.FTN77
Purpose:
To write a FEMGEN (or FAMbuild) input file from a geometry neutral file.
Author: Michael Mead, Rutherford Appleton Laboratory.
Size:

NF2FG.FTN77	:	134 kbytes
USERRD.FTN77	:	3 kbytes
SFOPEN.FTN77	:	1 kbytes

(Also requires RDNF)
Machine Dependencies:
1. Use of escape code in subroutine BELL;
2. Offset of 176 used when converting integers to character and back in subroutine I2CCON.
Development Status: Ready for use.
Restrictions:
Supports a subset of a modified CAD*I neutral file for CAD geometry, version 3.2.: only geometry levels 3a, 3b, 3c are allowed.
Some of the major modifications are;
1. scoping of entities is not recognised;
2. parameters of type LOGICAL and ARITHMETIC EXPRESSION are not permitted;
3. entities with scope have modified forms: they are not named.

B_REP models:

These arrive in FEMGEN as sets of surfaces, not as solid objects. Only EDGE_CURVEs of type LINE are permitted. Boolean operations are not possible.

POLY_HEDRON models:
These arrive in FEMGEN as sets of surfaces, not as solid objects. Boolean operations are not possible.

CSG models:
Primitive types LINEAR_SWEEP, ROTATIONAL_SWEEP and PLANAR_HALFSPACE are not permitted. Primitive types TRUNCATED_PYRAMID, REGULAR_PRISM, and TRUNCATED_CONE are not made into solid bodies. The SOLID_SPHERE primitive turns out an odd shape- this is being looked into. Boolean operations are not possible.

Language: Fortran-77

C.2.3 Software developed by GfS

C.2.3.1 Program NFUTIL

Purpose: Utility program for manipulating, copying, sorting and merging CAD*I files and metafiles.

```
input: any CAD*I  file / neutral file / metafile
output: a CAD*I  file / neutral file / metafile

features: list contents
          copy without change
          copy with blank and linefeed removal
          copy with pretty print
          switching between different files (enables merging)

version 2/88
language: FORTRAN77

size: executable  25 kB
      source      36 kB

machine dependencies: unit numbers for terminal in/output
                      program runs on any machine

status: distributed and tested

author: Helmut Helpenstein, GfS
```

C.2.3.2 Program FEREAD

Purpose: program for reading a neutral file with FEM input/output data connecting the common data base to GfS-FEM (pre-processor, computation part and postprocessor). Uses the scanner/parser program RDNF (see above).

```
features:
FEREAD can evaluate the following FEM input data:
 - nodes
 - discrete elements: springs, dampers, masses
 - 1D elements: bars, beams (general and profile)
 - 2D elements: shells, membranes, plates with 3 or 4 vertices,
                with or without additional nodes on edges
                and/or faces
 - 3D elements: hexahedrons, pentahedrons, tetrahedrons  with
                or without additional nodes on edges, faces
                and/or bodies
 - static loads
 - boundary conditions, degrees-of-freedom
 - material (isotropic)
FEREAD can evaluate the following FEM output data:
 - results of static analysis,
              eigenvalue analysis
              frequency response analysis
```

transient response analysis
- real or complex
- displacements
- stresses (component, principle, equivalent)
- strains (component, principle, equivalent)

input: RDNF data base for which RDNF had used
1. Keyword Definition File
2. CAD*I neutral file of FEM type containing FEM input and/or output data

output: FEM input/output data for GfS-FEM system in formats suitable for FE pre/post-processor TRANET

version 2/88
language: FORTRAN77

size: executable 100 kB
source 75 kB

machine dependencies:
- unit numbers for terminal in/output
- bit manipulation routines
- Ascii code
- record structure of data base

program is available for almost any machine.

status: distributed and tested

author: Helmut Helpenstein, GfS

C.2.3.3 Program FEINW

Purpose: Write FEM input data from GfS pre-processor PROLOG to a CAD*I neutral file (model and analysis part).

features: FEINW deals with
- nodes
- discrete elements: springs, dampers, masses
- 1D elements: bars, beams (general and profile)
- 2D elements: shells, membranes, plates with 3 or 4 vertices, with or without additional nodes on edges and/or faces
- 3D elements: hexahedrons, pentahedrons, tetrahedrons with or without additional nodes on edges, faces and/or bodies
- table of degrees-of-freedom
- material (isotropic, including dynamic and thermic properties)
- static point loads
- distributed loads (edge, surface, volume)
- preset displacements
- dynamic loads (steady state or transient)
- time or frequency ranges
- prescribed temperatures
- heat sources
- heat transfer

- ambient temperature
- analysis control for static, dynamic and thermic analysis
- substructure technique
- static condensation
- dynamic condensation
- output requests for displacements, stresses. strains

input: FEM input data from FEM pre-processor TRANET,
dialogue for analysis requests
output: CAD*I neutral file with FEM input data, i.e.
model and analysis part

limitations: max. 50 substructures allowed
max. 1024 elements per substructure

version: 2/88
language: FORTRAN77

size:	executable	74 kB
	source	128 kB

machine dependencies: - unit numbers for terminal in/output
- bit manipulation routines
program is available for almost any machine.

status: distributed and tested

author: Helmut Helpenstein, GfS

C.2.3.4 Program FEOUTW

Purpose: Write FEM output data (FE result part) to a CAD*I neutral file. To establish a proper relation between FEM input and results, FEOUTW reads the analysis part of a CAD*I neutral file; if this is not available, the user can specify the relation by dialogue.

features:
FEOUTW takes data from:
- static analysis
- real eigenvalue analysis
- complex eigenvalue analysis
- frequency response analysis
- time (transient) response analysis

It transfers the following data:
- displacements
- mode shapes (real or complex)
- eigenfrequencies
- stresses (component, principle, equivalent)

input: 1. FEM output data from GfS-FEM (GFSSAP or DYNA)
2. a neutral file with FEM input data (model and analysis part)

```
output: CAD*I neutral file with FEM output data, i.e.
        FERES part. On user's request this part can be
        appended to the input neutral file.
```

extendibility: Only one routine is responsible for the format of the results to be read. It can easily be changed or adjusted, thus linking results of other FEM programs to the neutral file.

version 1/88
language: FORTRAN77

```
size: executable   41 kB
      source       73 kB

machine dependencies: - unit numbers for terminal in/output
                      - bit manipulation routines
                program is available on almost any machine.

status:  distributed and tested

author:  Helmut Helpenstein, GfS
```

C.2.4 Software developed by ERDISA

C.2.4.1 Program CADMOD

Purpose: To transfer CAD models stored in a file of CAD*I format to an input file of the MSC/MOD FEM preprocessor (of McNeal Swendler Corp.).

Language: Fortran 77.

Size: Source 160 kB approx. Executable 920 kB approx.

Machine dependencies: Developed on a workstation with Unix (HP9000-300). MSC/MOD runs on a PC with MS/DOS.

Status: First version developed and tested.

Input: A neutral file of CAD*I format with CAD data.

Output: An input file for MSC/MOD of type "neutral".

Features: The CAD models that can be transferred are of type wireframe, surface, polyhedron, B-rep and CSG with the format defined in the CAD*I specification, version 3.2. MSC/MOD has the linear segment as its only geometric entity so CADMOD converts or approximates the CAD models into sets of linear segments. Some information is lost but it passes enough to make the FEM mesh using the MSC/MOD facilities. The parsing and scanning steps of this postprocessing are done by RDNF, written at RAL.

Restrictions: It does not consider the scoping of the entities. The parameters of type arithmetic expression are not permitted. B-spline curves and surfaces will be implemented in the next version.

Extendability: Modular development. Easy implementation of new entities. Easy adaptation to other FEM preprocessors of similar characteristics.

Authors: Luis Delgado and Jose L. Navarro, ERDISA.

C.2.5 Software developed by INITEC.

C.2.5.1 Program IENF

Purpose: The Insert Eproperty Neutral File program (IENF) has been conceived as an auxiliary program in the interface between CADD-FEM, FEM preprocessor of Computervision, and the CAD*I Neutral File. It facilitates the insertion of element properties in the computervision data base according to the CAD*I specification.

Language: NEWVAR (Computervision special language to access its data base).

Size: executable 7kB. Source 6kB

Machine dependencies: CADDS-4X environment with CGOS-200 Operating System.

Status:First version developed and tested.

Input: Parameters of the new element property interactively provided by the user who is addressed by the menus of the program.

Output: The CADDS-FEM internal data base stores a new EPROPERTY of CAD*I type. This follows the special CADDS-FEM philosophy to handle this kind of data.

Extendability: New element properties defined in the CAD*I specification can easily be included in the modular program where the first steps in the addressing menu are common.

Author: Javier Moldes, INITEC.

C.2.5.2 Program CVNF

Purpose: Enables the transfer of a FEM model created by the Computervision CADDS-FEM preprocessor to the CAD*I Neutral File.

Language: Formatter (Computervision specific language for data base reading).

Size: Source 17kB

Machine dependencies: CADD-4X environment with CGOS-200 Operating System

Status: First version developed and tested.

Input: CADDS-FEM data base with the model and some data provided by the user by means of menus.

Output: The CAD*I Neutral File of the WORLD and FEM model required.

Extendability: New data to be passed to the Neutral File from the model data base will require new subroutines.

Author: Javier Moldes. INITEC.

C.2.5.3 Program NFAN

Purpose: This program reads a neutral file of FEM data and produces the corresponding ANSYS input file.

Language: FORTRAN 77.

Size: Source 180 kB approx. Executable 220 kB approx.

Machine dependencies: The first version has been developed in a PC-AT with MS/DOS.

Status: First version developed and tested.

Input file: ASCII file of FEM data in CAD*I ver 3.1 format.

Output file: An input file of ANSYS.

Features: This first version only transfers FEM data for static linear analysis.

Extendability: Next versions will consider the data for modal and dynamic analysis.

Author: Javier Moldes, INITEC.

Discussion of the CAD-FEM Data Transfer

D.1 Introduction

Computer Aided Design (CAD) and the Finite Element Method (FEM) are very different fields. Their respective objectives are:

- Designing the shape of objects by means of geometrical data

- Modelling the physical behaviour of objects under some given conditions, (physical means: mechanical, thermal, etc). The geometry is only a part of this modelling. It is very important but not the essential matter.

The geometry is the common basis of these two techniques. Geometrical data are used in FEM. In industry, FE analysis results validate a CAD design or they imply changes to it. This requires exchanging data transformed to fit for each technique. These transformations are currently made by experts in a way that is almost an art. The FEM experts use some rules that come from their knowledge, experience and intuition. These are the "Heuristic rules of the CAD/FEM data transfer".

The geometric design and FEM techniques are computer aided. The exchange of data between them requires software interfaces between packages. The automation of this interfacing leads to the inclusion of rules in the interface software. Regarding this objective the heuristic rules currently are influenced too much by the human element. This task of the CAD*I project is intended to make some initial considerations about how these rules might be developed in an objective manner and included into the interfaces.

D.2 CAD/FEM interchange environment

The CAD and FEM techniques are parts of the whole design procedure of a product. In this procedure the following phases can be considered [1]:

- Collection of Requirements: performance, physical dimensions, loads, environmental conditions, etc.
- Overall design sketch: process description, concepts, external geometry, etc.
- Assembly structure: materials, components, modules, interfaces, etc.
- Detailed design of new components: Detailed geometry of components, load cases, analysis, results, verification, etc.
- Evaluation of prototypes: performance test results, dynamic analysis, experimental verification, suggested changes, etc.

CAD and FEM participate and are influenced by several factors from each one of these phases.

The FEM part itself consists of the following phases:

- Preparation of data: dimensioning criteria, external influences, functional description, design geometry and expected/estimated results.
- Input data: element distribution and connectivity, element type and geometry, material properties, boundary conditions, loads, analysis cases and required output data.
- Calculation: Displacements, stresses, eigenfrequencies, etc.
- Evaluation: accuracy of the results, comparison with the expected results, evaluation of the assumptions of the preparation and input data phases, etc.
- Action: The results are translated to design suggestions and new analysis may be necessary.

These phases are illustrated in Figure 15.

The CAD/FEM transfer is influenced by the five phases of FEM but it has most importance in the preparation and input data phases [2]. In these the FEM preprocessor programs are currently used. The basic data manipulated by these programs are geometric, as the CAD data. The geometric data for FEM preprocessing is available either as

- a traditional drawing, (on paper media), or
- as a CAD data set, (on a magnetic tape or similar media)

The main subject of the CAD/FEM data exchange is the transfer of the CAD data set into the geometric FEM preprocessor data. However, from this brief explanation of the whole environment, it is obvious that a FEM model includes many other factors besides the geometry. These also determine the criteria to be taken into account in the development of the CAD/FEM heuristic rules.

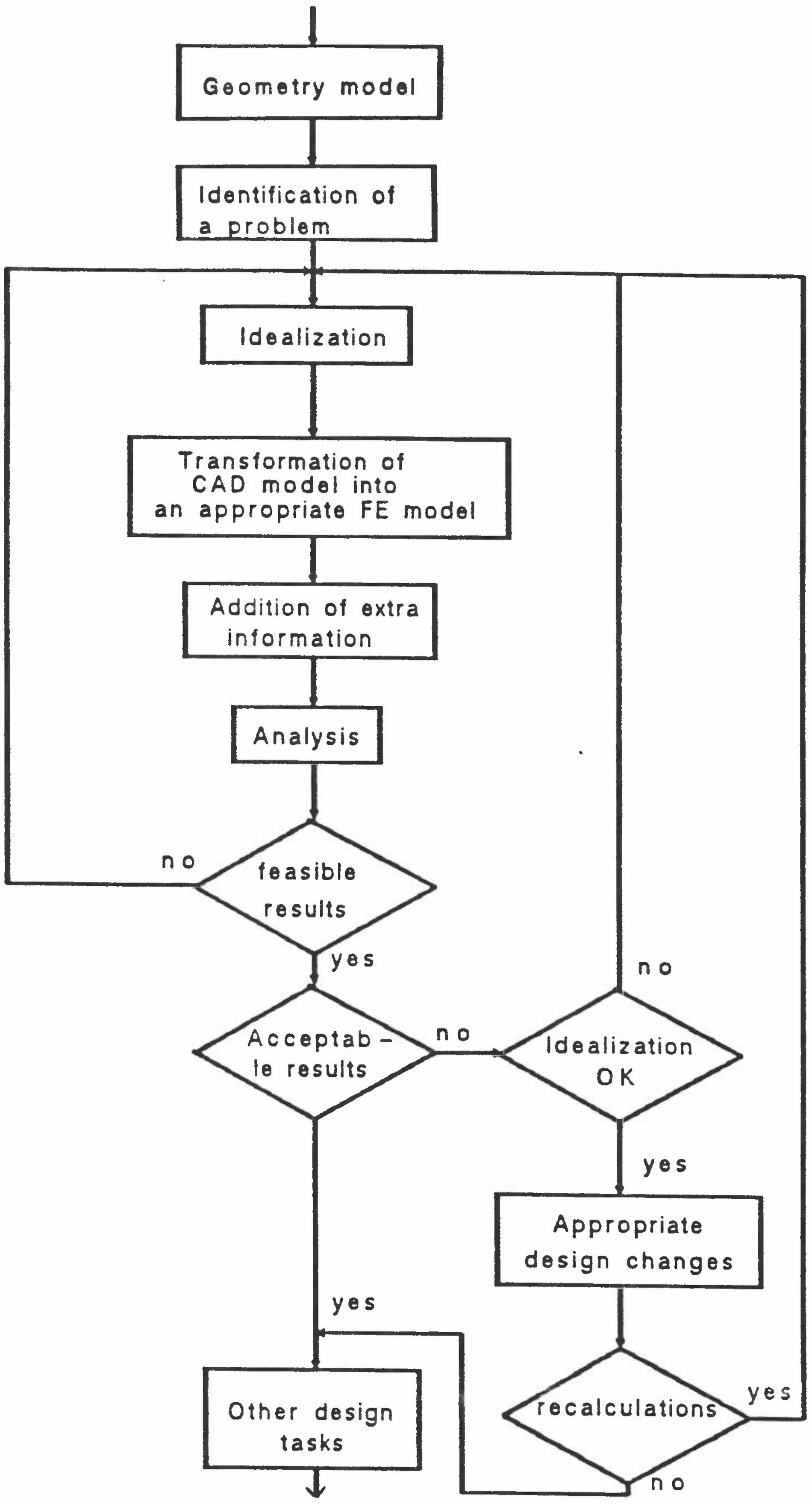

Geometry model
Identification of a problem
Idealization
Transformation of CAD model into an appropriate FE model
Addition of extra information
Analysis
no
feasible results
yes
Acceptab - le results
no
Idealization OK
no
yes
Appropriate design changes
yes
recalculations
yes
no
Other design tasks

Figure 15. Connection between Modelling and Analysis.

D.3 The CAD*I approach to the CAD/FEM interface

The CAD to FEM interface has already been approached in the CAD*I project [1 through 7]. It has been made considering two basic data references:

- The CAD*I neutral file as the source data of CAD models, and
- the "FEM preprocessor" programs to make all the FEM model preparation.

Three conceptual phases are considered to convert the CAD geometrical data into FEM suitable data, these are:

- Geometric idealisation,
- mathematical transformation of geometric entities,
- discretisation of the model (mesh generation).

The geometric idealisation means the simplification of the complete CAD model to be used by the FEM preprocessors. This is based on the approximate nature of the FEM method. Different kinds of transformations of geometry can be made to obtain the idealised model, all of them require human intervention:

- removal of unnecessary details,
- idealisation of geometric dimensions,
- elimination of a part that is symmetrically equivalent to another.

The mathematical transformation of geometric entities means the transformation of CAD entities into other equivalent and simpler ones which are understandable to the FEM preprocessor. These transformations can be made automatically by means of currently available software and some interfaces have already been developed in the CAD*I Project. These apply mathematical and geometrical transformations to the CAD entities of the CAD*I neutral file to obtain the equivalent ones in different FEM preprocessors, e.g.:

- NF2FG: interface to FAMbuild of FEGS Ltd., written at RAL.
- GEOREAD: interface to the GFS-PROLOG of GfS mbH, written at GfS.
- NEH61: interface to PIGS of PAFEC Ltd, written at NEH.
- CADMOD: interface to MOD of MSC Corp., written at ERDISA.

With respect to the discretisation of the model, the mesh generation can be done by modern mesh generating tools almost automatically. However, the mesh is not the only step to reach a FEM model; as was shown in the previous chapter, many other factors have to be added to this mesh to have a model which will simulate the behaviour of the product under some given set of conditions. These other factors and their influence on the geometric shape of the object cannot be dealt with automatically. The human intervention to complete the FE data cannot be avoided. Very often, the suitable FE model is not produced in the first attempt. The first results of the analysis may show that modifications are necessary.

The division of the CAD/FEM transformation flow into these three phases can give the impression that it can be easily structured and programmed. This is not true because human intervention guides the process from the beginning to the end; the experts have in mind all the phases of the whole process and its environment. The model idealisation is affected by the mesh that will be used and this depends on the loads, boundary conditions and the other additional factors that are not present in the CAD model; also the experts' background experience can introduce new considerations almost "intuitively", and other ones that come from economical factors.

To reach a FEM model starting from a CAD one can be quite difficult; although the CAD*I project has produced some CAD to FEM interfaces they are limited to a particular aspect of the FEM model construction. Human intervention, using the heuristic rules of the experts, is still essential to handle other aspects.

D.4 Guidelines for FE modelling

D.4.1 The complexities involved

Most finite element practitioners would agree that designing an efficient finite element model for a given problem is as much an art governed by intuition as a skill acquired by experience. Significantly, the most common treatises on FE technique [8 through 12] do not include any section specifically devoted to defining general rules or guidelines for this purpose of developing efficient FE models; only Taig and the NAFEMS institution [13] have published a set of guidelines to the FE practice in general, that includes some subjective rules to take into account within FE modelling.

Of course, there is a general rule that the finer the mesh the more accurate the results, but it could lead to prohibitively high computation costs, even if a solution could be found with the computer power available.

In order to come up with a cost effective FE model the experts weigh, among others, the following factors:

a) the requirements of the solution,

b) the nature of the problem,

c) the mechanics of the problem,

d) the geometry or domain where the solution is sought,

e) the applied loads (or excitation functions) and constraints,

f) the analysis computer code.

However, these factors are not to be considered independent of one another, as they are highly interrelated. These factors are to be taken into account throughout the CAD to FEM data transformation.

A brief explanation on each one of these factors is given in the following sub-chapters.

We shall speak in terms of displacements, stresses or forces as in the more conventional structural analysis. The corresponding terms in thermal or fluid mechanics should be understood.

D.4.1.1 The requirements of the solution

In most cases the goal of the finite element analysis is to validate a design, and the designer has previously come up with a gross estimate of what the safety margins are. If the design margins are expected to be considerable, there is no need for a very detailed model: a coarse model, as long as it is conservative, might do. The analyst must remember, however, that finite elements normally yield solutions that are approximate on the unconservative side, so enough conservatism must be introduced in the model by some other means (neglecting to account for certain material contributions, conservative boundary conditions, overestimating the loads, etc.).

The opposite happens when the design margins are very narrow. In this case the sources of conservatism must be carefully trimmed down, and the model needs in turn to be very accurate so as not to smooth the stresses and miss the peaks. One must keep in mind that the stress results for an element are some sort of average values of the actual stresses within the element.

D.4.1.2 The nature of the problem

The analyst in charge of designing a FE model for a problem normally knows beforehand the type of solution to be performed and designs the mesh accordingly. A thermal problem, for example, is a well-behaved problem that smooths out discontinuities. Some problems in fluid mechanics, on the other hand, have a tendency to generate discontinuities in the solution that cause large inaccuracies throughout the domain, unless the grid is well refined in the affected areas. Also, whether the problem is static or dynamic has important consequences on the modelling as we shall see in the next subsection.

D.4.1.3 The mechanics of the problem

For a given problem, it is of advantage to assess previously how the individual components of the system will contribute to the overall solution. For example, if a structural engineer knows that a shell component of a structure will be not subjected to significant bending in regions far from the supports, then he will use large elements there where the stress fields are expected to be of membrane type, and will concentrate small elements near the supports in order to capture the local bending in those regions.

Sometimes the critical areas can be easily pinpointed. Then the mesh needs to be refined only in those regions, and a coarse grid might be sufficient for adequately modelling the stiffnesses of the remaining parts of the system.

The influence of the boundary conditions must also be considered. In fact, sometimes the boundary conditions are the cause that complicates the problem to the point of requiring numerical modelling. The first difficulty with the boundary conditions frequently arises from the fact that they are not easy to define even for the mathematically exact model. Once the mathematical boundary conditions are established, they need to be discretised. This discretisation is very much problem-dependent.

For static problems, the boundary conditions usually affect only the local areas. The situation is very different for dynamic problems. In these, the boundaries cause reflections and refractions in the wave propagation patterns that significantly affect the response of the system. Adequate modelling of the actual boundaries and the artificial ones is essential in this case.

D.4.1.4 The geometry

The geometry of the region has a definite influence in the sizes and types of the finite elements to be used in the model. However, in most cases it is not required that the finite element model be a detailed replica of the component under analysis. What counts, after all, is that the mechanical behaviour of the component be modelled faithfully. For this reason sometimes whole portions of the system can be either totally eliminated from the model or represented by simple elements like springs and masses. A knowledge of the substructuring techniques is of value to reduce the size of the problem.

The geometric idealisation of the system may be much simpler than the actual geometry, but this idealization must retain the essentials of the mechanical behaviour. Discontinuities are always a source of concern. Holes, for example, always have to be considered due to the stress concentrations that occur around them. Only when small and located away from the critical areas the holes may be omitted in the model and their influence considered a posteriori by applying a stress intensification factor to the computer results. Re-entrant corners or edges are another source of errors in the approximation. Szabo [14] describes how it may be more effective to round the sharp edges with cylindrical or spherical fillets than to refine the mesh around.

When the component under analysis has symmetry properties the size of the problem can be greatly reduced. For linear problems, asymmetric loads can be dealt with by superposition of symmetric and asymmetric loading conditions. It is necessary, however, to make separate analysis runs for the two sets of boundary conditions, and, frequently, to double the number of loading combinations. For small problems these burdens may offset the advantage of the reduced computation cost.

For dynamic problems, it is often important to make use of the symmetry properties. Otherwise round-off errors may result in several closely spaced vibration frequencies where the real object has only one.

D.4.1.5 The loads

The type of loads acting on the model has to be considered when designing the finite element mesh. Concentrated loads normally require refined meshes, due to the steep stress gradients that the solution has to capture. Well distributed loads over large portions of the structure, on the other hand, normally allow for coarser meshes. One has to remember, also, that concentrated loads will appear at the boundary nodes.

In addition, the nature of the load, static or dynamic, also has an influence on the modelling. In dynamics it is important that the mesh size be a fraction of the wave length for the numerical wave to propagate correctly. In dynamic problems the masses, in addition to the stiffnesses, have to be properly distributed. If the load frequencies are close to the resonant frequencies of the system it is important to determine how small changes in the model affect the model frequencies, to avoid overlooking resonant conditions.

D.4.1.6 The computer code

Finally, the library of elements available in the code to be used for the analysis has an obvious influence in the design of the finite element mesh. If the library contains high order elements, the analyst has the freedom to decide on whether to use large high order elements or small simple elements. Otherwise the choice is limited. This choice of h- or p-elements with a coarse or refined mesh is never obvious and must be considered on a case by case basis [15].

A thorough understanding of the capabilities and peculiarities of the code library elements to be used in the model is essential.

* * *

The above discussion represents an effort to classify the rules for finite element modelling in six groups or sets, however interrelated. It also reveals the complexity of the task of defining heuristic rules to help the practitioner develop efficient finite element models. Currently, however, a lot of effort is being spent for this purpose, as is shown in the next section.

D.4.2 Advanced FE Modelling.

D.4.2.1 The state of the art of automatic FE modelling

In recent times the advent of the microcomputer has spread the use of finite elements into every corner of industrial manufacturing and design and into every field of research and development. This phenomenon, in turn, has caused the development of a large number of finite element computer programs, some of very general applications, others orientated to very specific tasks.

This abundance of finite element codes and the large number of users have given rise to two types of concerns in the scientific community:

1) the quality assurance of the circulating computer programs, and,

2) the user qualification to operate the codes with knowledge, in view of the complexity of the object.

The scientific community is addressing the concern on the reliability of commercial finite element codes by issuing calls for the development of standards in the performance of codes [16] and of the tests to subject them to [17]. Calls are also made for standard methods of testing individual finite element models [18] such as requesting them to pass the Iron's [19] patch test [20].

The concern about finite element user's qualification is being aggressively addressed by both the research community and industry. The quantity of information required to assimilate for efficiently using finite element codes amounts in many cases to thousands of pages. It is recognized, moreover, that this information alone will not provide the user with the skills that are normally acquired only through insight and experience. For this reason a lot of work is currently being done towards the development of expert systems that bring together the knowledge and experience of finite element experts and make them available to the practitioner.

The expert system sought will help the user decide on the kind of the analysis to be performed, the proper idealization of the geometry, the discretisation parameters, the types and shapes of the elements to employ, the anticipated cost of the run and the accuracy to expect from the model. It will also monitor the progress in the development of the model, will offer advice when requested and will reason about its own advice. In the next step the expert system will be able to acquire information on the geometry of the mechanical body from a CAD/CAM file and build the finite element model with minimum user assistance.

Such a goal is not far-fetched. Actually, if such an expert system is not available yet on a commercially useful basis, many prototypes have already been developed and are being tested for the limited purpose of offering assistance to the user.

Table 1 is a compilation of the various prototype expert systems described in the literature. The best description of what one such prototype encompasses is perhaps that of FEMOD [21], which is also the latest reference found on the subject.

The most basic expert system consists of two modules [22]: a knowledge base and an inference engine. The next upgrading is obtained by adding an explanation facility that allows the system to reason about its own decisions. The most sophisticated expert systems also incorporate a knowledge acquisition facility as a fourth module.

Typically, an expert system is built with the help of a commercial shell. These expert tools provide an empty knowledge base, a domain independent inference engine and offer a user friendly interface for the builder to develop the knowledge base [13].

TABLE 1

EXPERT SYSTEMS FOR FE MODELLING FOUND IN THE LITERATURE

NAME	YEAR [Ref.]	FEATURES
SACON	1978 [23]	.- For users of the MARC code
(Not given)	1980 [24]	.- For USA Air Force
ESSDAN	1985 [25]	.- For users of the FIPTIP code
SPERIL-I and II	1985 [26]	.- Structured damage assessment
HI-RISE	1985 [26]	.- Design of hi-rise buildings
FEASA	1986 [27]	.- Incorporates an explanation facility
ADEPT	1986 [28]	.- To be integrated with a CAD solid modelling system
(Not given)	1986 [29]	.- Optimisation based on a weighted error computation .- Incorporates empirical data base for standard subregions
(Not given)	1987 [15]	.- Incorporates an explanation facility .- Optimisation based on a-priori error estimation .- For hp-elements
FEMOD	1988 [21]	.- Knowledge acquisition facility under development .- For users of the EAL code
ESA	1988 [30]	.- For nonlinear analysis

D.4.2.2 An expert system for the CAD/FEM interface

From the last chapter, it can be concluded that a CAD/FEM interface that incorporates the heuristic rules has to be an expert system. Some expert systems have been developed along more or less the same lines. To accomplish a good expert system, the most fundamental part is to define the rules that will be incorporated into the knowledge base. As is known, this is not a trivial point but the most difficult part of such systems. The other part (inference engine) is already available on the market as a standard piece of software.

To build the knowledge base of an expert system usually requires some FEM experts and AI experts working together for a long period, in order to express the human intuitiveness and skill as objective rules (often with a weight or probability factor assigned to each one), structure them into levels and hierarchies, and determine the non-structurable interconnections between them if required.

The next chapter will attempt to offer a possible model of the knowledge base for FE Modelling. It does not try to be exhaustive but to present a possible method to form the first levels of a tree of decision rules and show how this tree could integrate some FEM heuristic knowledge in a network of objective rules, which appear mutually influenced almost from the root levels.

To increase the precision of such a knowledge base, the concepts and their connections must be deepened from this superficial level down to almost the levels that belong to the intuition of the experts. But we judge that this deepening can be useful before becoming infinite.

Other aspects of the expert system implementation are of minor importance but they could connect the system to other tasks of the CAD*I project, as for instance:

- The additional factors to build the model can be provided to the expert system interactively, by means of a dialogue with the user addressed by the system. Some of them could be included in the original CAD model. This would reduce the human intervention. To have this data together with the geometric model, the CAD system should include advanced features (similar to those being studied by WG5 of CAD*I) which would allow the inclusion of loads, boundary conditions, and other factors as part of the CAD description.

- The expert system should be completed with a module for automatic mesh generation on 3D structures as part of a complete interface, or it could just pass some instructions to a classical FEM preprocessor program. The automatic mesh generation, following user's instructions, is already provided by some FEM preprocessors, in the most advanced ones only the mesh density has to be provided by the user.

- Successful expert systems were achieved by a reduction of the scope of the matter into well defined sub-fields. The first step on this way could be to separate the human expertise fields (mechanical, static, dynamic, modal, thermal, etc) in different modules.

- Some mechanism of reference from the geometric parts of the FE model back to the original CAD ones have to be provided to allow for recovering the original data during the modelling approximate cycles.

D.5. Heuristic rules for FE Modelling

The term heuristic is used in the theory of education as an adjective regularly applied to the rules that a learner should discover for himself. It is also used as a name to describe the method of solving problems by inductive reasoning, by evaluating past experience and moving to a solution by trial and error.

As described before, expert systems consist, basically, of an inference engine that accesses a knowledge base. This knowledge base contains a large number of heuristic rules that may be applied at any one time to solving the problem at hand. While the inference engine is problem independent, and many are commercially available, the kernel of each expert system is its own knowledge base.

If the rules that apply to a particular knowledge are essentially heuristic, as most frequently they are, the way to formulate such a base knowledge is to enlist the help of a domain expert. By the very heuristic nature of the rules, however, it turns out to be very difficult to formulate the knowledge base in a systematic way. Recent advances in Artificial Intelligence are making it possible to develop knowledge acquisition facilities to the expert systems that enable them to learn by experience, as the domain experts do.

D.5.1 Development of a model of heuristic rules set

In this section we will propose heuristic rules that apply to the domain knowledge of finite element modelling in continuum mechanics. The methodology will be such that each rule applies a certain weight factor to the output variables.

The output variables for the problem of finite element modelling are taken to be the degrees of mesh refinement for the various regions of the model. The way a model is to be subdivided into regions will also be a subject of heuristic reasoning. The final weight for each variable will determine the starting overall mesh density required for the particular region of the model. With this information a potent mesh generator may be called into action to develop a possible mesh pattern.

The method described below has the advantage of being very general, and suitable to be incorporated into a knowledge acquisition module. As adaptive mesh refinement codes are widely available today, it appears to be at least conceptually simple to develop a statistical data base to optimize the weights originally assigned to each output variable by each of the heuristic rules invoked. This point will be reconsidered later.

D.5.1.1 Heuristic Rule Set 1 (HRS1): Model regions

A general model will have regions of the following types:

- areas where concentrated loads are applied or other conditions are present, such as geometry discontinuities, boundaries, etc. which make them candidates for stress concentrations (Region 3),

- areas where stress gradients are expected to be moderate (Region 2),

- areas not included above with distributed loads applied (Region 1),

- areas with none of the above conditions (Region 0).

The model is thus divided in four regions where the output variable (mesh refinement) will take on different values. In general, the higher the number assigned the more complex the solution will be in that region and, consequently, the mesh will be more refined there.

An "a priori" determination of these regions may not be simple, as a certain degree of expertise may be required. If automation of this task is required, it will have to come from preliminary analyses that detect the problem areas. (see Heuristic Rule Set 2, level 0).

D.5.1.2 Heuristic Rule Set 2: Required overall accuracy

- Gross estimate suffices or preliminary analysis (0),
- standard accuracy (1),
- refined accuracy (2),
- high accuracy required (consistent with the limitations of the finite element technique) (3).

Each one of the above requirements will impose a degree of refinement in every region of the model. This is accomplished by the weights that each requirement assigns to the output variable (mesh refinement index) associated with each region. One proposed set of weights (dimensionless) is shown in Table 2.

TABLE 2

Weights for modelling refinement assigned by Heuristic Rule Set 2

	Required overall accuracy			
Regions of Analysis	0	1	2	3
0	1.	1.	1.	2.
1	1.	2.	2.	3.
2	2.	3.	4.	5.
3	3.	5.	8.	10.

D.5.1.3 Heuristic Rule Set 3: Type of analysis

- Linear, static (time independent) (0),
- nonlinear static (1),
- linear dynamic (time dependent) (2),
- nonlinear dynamic (3).

The proposed weights are listed in Table 3.

TABLE 3

Weights for modelling refinement assigned by Heuristic Rule Set 3

	Type of analysis			
Regions of Analysis	0	1	2 (*)	3 (*)
0	1.	1.	1.	1.
1	1.	1.	1.	1.
2	1.	2.	1.	2.
3	1.	3.	1.	3.

(*) Use of the Guyan reduction technique may be required

D.5.1.4 Heuristic Rule Set 4: Type of problem

- Heat transfer (0),
- structural mechanics (1),
- fluid mechanics (2).

The proposed weights are listed in Table 4.

TABLE 4

Weights for modelling refinement assigned by Heuristic Rule Set 4

	Type of problem		
Regions of Analysis	(*) 0	1	2
0	1.	1.	3.
1	1.	2.	3.
2	2.	3.	5.
3	3.	5.	7.

(*) Regions of steep temperature gradients should be understood here

D.5.1.5 Heuristic Rule Set 5: Geometry type

- Assembly of regular 1D(*) elements (0),
- assembly of 2D(*) flat elements (1),
- assembly of 2D(*) non-flat elements (2),
- assembly of 3D(*) elements (3).

(*) It is recognised that elements will always be three-dimensional. What we are trying to evaluate here is their suitability to be represented by one-dimensionally dominant elements like slender prisms (beams) or by thin plates (considered 2D elements). Otherwise the complexity of the structure requires modelling the behaviour independently in all three dimensions.

The weights assigned in each case are listed in Table 5.

TABLE 5

Weights for modelling refinement assigned by Heuristic Rule Set 5

	Geometry type			
Regions of Analysis	(1) 0	(2) 1	(2,3) 2	(4) 3
0	1.	1.	2.	1.
1	1.	2. - 4.	4. - 8.	1.
2	1.	4. - 8.	8. - 16.	2.
3	1.	4. - 8.	8. - 16.	3.

(1) 1D elements are exact and therefore require no additional refinement.

(2) Typically, the use of triangular elements will require twice as much refinement as the use of quadrilateral elements. Shapes and aspect ratios must also be considered.

(3) Double weight is assigned here for non-flat elements. This value, however, should depend on the actual degree of warping in the element.

(4) 3D element refinement cannot be increased at leisure due to the high number of degrees of freedom of each of these elements.

D.5.1.6 Heuristic Rule Set 6: Geometry complexity

This set of rules somehow overlaps with HRS5 and is also related to HRS1. It is however considered independently in order to minimize the complexity of the work. In the most general case, this set of rules should interact with HRS6 to define a three dimensional matrix of weights.

- Highly regular distribution of elements; no singularities (*) present (0),

- few singularities (*) present (1),

- many singularities (*) present (2),

(*) By singularities we mean complex details, boundaries, re-entrant corners, etc.

The associated weights are listed in Table 6.

TABLE 6

Weights for modelling refinement assigned by Heuristic Rule Set 6

	Geometry complexity (**)		
Regions of Analysis	0	1	2
0	1.	1.	1.
1	1.	1.	2.
2	1.	2.	4.
3	1.	4.	8.

(**) In many cases the complexity of the geometry will impose rather than require the degree of mesh refinement. One consideration should be given to whether small geometrical details must be represented or could be smoothed.

D.5.1.7 Heuristic Rule Set 7: Load time distribution (applicable to dynamic analysis only)

The effect of the load spatial distribution was already taken into account when the model regions were defined. However, the definition of the model regions defined for static analysis must be revised on account of the following.

In dynamic analysis the masses become sources of loads. As a consequence, when defining model regions based on external loads the distribution of masses must be taken into account. Concentrated masses will give rise to concentrated loads, etc. However, the need for accurate modelling of the masses is not as acute as for the modelling of stiffnesses.

Besides the load spatial distribution, the discretisation in time must be considered. In dynamic analysis the load time distribution affects primarily the part of the analysis concerned with the time dimension parameters, which is not considered here. However, it affects the geometrical model as well. If waves are generated that propagate with a velocity c (which is a function of the elastic constants), in order to capture the wave propagation patterns properly it is required that the spatial discretisation mesh size, h, be such that h<c*Dt, Dt being the time increment used in the discretisation analysis.

This effect is independent of the model regions and the consequence of it is that for region 0, that of mimimum discretisation refinement, the size of the element be less than c*Dt, for the smallest Dt considered in the analysis.

This heuristic rule will be implemented by requiring a maximum element size in all regions of the model.

D.5.1.8 Heuristic Rule Set 8: Program analysis capabilities

Many programs nowadays combine the h- and p-element capabilities. The p-element allows for the use of higher order interpolation functions. In general, a p-element will require a lower degree of mesh refinement, proportional to its interpolation order.

When the program to be used in the analysis incorporates p-elements, the user has the choice of reducing the mesh refinement by increasing the interpolation order accordingly.

The effect of this capability would be to apply a weight factor less than 1 there where the interpolation order is increased.

D.5.2 Suggestions on how the heuristic rules might be implemented in an expert system

In the preceding subsection we have tried to identify the various sets of heuristic rules that govern the design of a finite element model. We have also proposed a methodology for a first order approximation to the problem of quantifying the impact of the various heuristic rules on the final model. This methodology consists of correlating the rules with some weight factors assigned to the output variables, which are the mesh refinements at each model region.

In the procedure proposed the effect of each set of heuristic rules has been decoupled from the rest. The matrix of heuristic rule interdependence has been diagonalised. In a more accurate approach this interdependence of heuristic rules must be included. The following discussion is an attempt to anticipate how this can be accomplished.

Finite element codes exist today which include the adaptive mesh refinement capability. What the program does is to evaluate a certain norm of the error in the approximation within each finite element. Where the error estimate exceeds a certain threshold the program automatically refines the interpolation. This can be accomplished by subdividing the mesh (h-elements), increasing the interpolation order (p-element), or both. This type of finite element code may be a basic tool for 'breaking in' of expert systems, as envisioned next.

In a first set of numerical experiments, the code with adaptive mesh refinement capabilities may be used to come up with more realistic estimates of the weights proposed in tables 2 through 6. By changing the conditions that govern the heuristic rules one at a time and evaluating the program final mesh one could determine the appropriate weights.

The interdependence of sets of heuristic rules may be evaluated by changing two, or more, conditions simultaneously. By performing enough numerical experiments a statistical data base could be established that allowed for certain functions to be interpolated which could, in turn, be used in the future for evaluating output weights taking into account the interdependence of the heuristic rules.

D.6 The FEM to CAD interface

This other side of the complete product design cycle has been studied in less depth that the previous one. This does not mean the usual FE postprocessing tasks where the FE results can be translated into graphics. The geometric deformations of the object are not useful to the CAD design directly. The FE results answer to the physical behaviour of the complete system that was modelled, and they are expressed in terms of values of physical variables, not just geometry.

How to translate the analysis results into modifications of the CAD design is something made by human experts. Only the optimisation techniques, that some FEM packages include, can be considered a first help to them. Optimisation in FEM means that some of the input variables of the FE modelling are left as parameters without a concrete value. Therefore the results are expressed as a function of these parameters and, knowing the required behaviour of the product one can estimate the optimum values of the parametrised variables.

Optimisation is not enough to make automatically the way back from the analysis results to the geometric design. Again the experts are indispensable. Their knowledge, experience and intuition determine which variables have to be parametrised or how the rough results have to be interpreted. Also, the initial conditions and general environment of the product design intervene here. A deep study of the relations between these data will help to develop tools to make the FEM/CAD interface more or less automatic. The impression is that the expert system technique will also have to be used in this other side of the cycle.

D.7 Conclusions

The report describes an approach to the CAD/FEM interface with a possibly automatic FEM modeller which takes its basic input data from a CAD geometric model. Many factors have to be taken into account besides the sole geometry. They are currently handled by human experts who use heuristic rules, i.e., rules from their knowledge, experience, and intuition. The interrelations among these factors, and the transformations of the geometric model that they mean, are very complex and the classical software does not seem useful.

The expert system technique appears to be the only means to include heuristic transformations in an interface between CAD and FEM programs. Some expert systems have already been developed, a general review of the state of the art of this matter has been shown.

A structured set of heuristic rules for FE modelling and a method to include them in a future intelligent interface have been offered as an example. This represents a first view of a knowledge base. Achieving a more complete description would require a team of FEM and AI experts to work intensively on this problem.

References

[1] T. Ladefoged and A.M. Spliid, "Scheme for a Report Concerning the Task Heuristic Rules of the CAD/FEM Interface," report CAD*I.WG6.NEH.005.88, NEH, Broendby, Denmark. (1988).

[2] C.J. Petersen, H. Kullmann, B. Palstrom and A.M. Spliid, "Design - Finite Element Analysis - Design," report CAD*I.WG2/WG6-7.NEH.001.86, NEH, Broendby, Denmark. (1986).

[3] T. Ladefoged and A.M. Spliid, "Relations Between CAD Models and FEM Models," report CAD*I.WG6.NEH.010.87, NEH, Broendby, Denmark. (1987).

[4] M. Peralta and L. Delgado, "Approaching the CAD/FEM Interface," report CAD*I.WG6.ERDISA.003.87, ERDISA, Madrid, Spain. (1987).

[5] M. Peralta and L. Delgado, "Geometric Features of Current FEM Preprocessors," report CAD*I.WG6.ERDISA.002.87, ERDISA, Madrid, Spain. (1987).

[6] M. Mead, "Interfacing Geometry to FEM Using CAD*I. Examples of Practical Problems," report CAD*I.WG6.RAL.009.87, RAL, Chilton, Didcot, Oxon, U.K. (1987).

[7] H. Helpenstein, "Some Rules for Data Transformation from CAD to FEM," report CAD*I.GfS.WG6.007.88, GfS, Aachen, F.R.G. (1988).

[8] O.C. Zienkiewicz, "The Finite Element Method," 3rd ed., McGraw-Hill, 1977.

[9] R.H. Gallager, "Finite Element Analysis Fundamentals," Prentice-Hall, 1975.

[10] K.J. Bathe, "Finite Element Procedures in Engineering Analysis," Prentice-Hall, 1982.

[11] B. Irons and S. Ahmad, "Techniques of Finite Elements," Ellis Horwood, 1980.

[12] T.J.R. Hughes, "The Finite Element Method. Linear Static and Dynamic Finite Element Analysis," Prentice-Hall, 1987.

[13] I.C. Taig, "NAFEMS - Guidelines to Finite Element Practice," Department of Trade Industry, National Engineering Laboratory, East Kilbride, Glasgow, Scotland. (1984).

[14] B.A. Szabo, "Geometric Idealizations in Finite Element Computations," Communications in Applied Numerical Methods, Vol.4, 393-400 (1988).

[15] E. Rank and I. Babuska, "An Expert System for the Optimal Mesh Design in the hp-Version of the Finite Element Method," International Journal for Numerical Methods in Engineering, Vol. 24, 2087-2106 (1987).

[16] W.M.Mair, "The Objectives of the National Agency for Finite Element Methods and Standards," Computers and Structures, Vol.21, 875-879 (1985).

[17] R.H. Macneal and R.L. Harder, "A Proposes Standard Set of Problems to Test Finite Element Accuracy," Finite Elements in Analysis and Design, Vol. 1, 3-20 (1985).

[18] H.D. Hibbit, "Some Issues Associated with the Validation of Finite Element Analysis," Finite Elements in Analysis and Design, Vol. 2, 119-124 (1986).

[19] B.M.Irons, "Engineering Applications of Numerical Integration in Stiffness Methods," AIAA J., Vol. 4, 2035-2037 (1966).

[20] R.L. Taylor, J.C. Simo, O.C. Zienkiewicz and C.H. Chan, "The Patch Test - A Condition for Assessing FEM Convergence," International Journal for Numerical Methods in Engineering, Vol. 22, 39-62 (1986).

[21] J.L. Chen and P. Hajela, "FEMOD: A Consultative Expert System for Finite Element Modelling," Computers and Structures, Vol. 29 99-109 (1988).

[22] C.L. Dym, "Expert Systems. New Approaches to Computer Aided Engineering," Engineering with Computers, vol. 1, 9-25 (1985).

[23] J. Bennett, L. Creart, R. Englemore and R. Melosh, "SACON: a Knowledge Based Consultant for Structural Analysis," Technical report STAN-CS-78-699, Standford University (1978).

[24] J.M. Rivlin, M.B. Hsu and P.V. Marcal, "Knowledge Based Consultation for Finite Element Structural Analysis," U.S Air Force Flight Dynamics Laboratory Report AFWAL-TR-80-3069, Wright-Paterson Air Force Base, Ohio (1980).

[25] S.C.-Y. Lu, "A Consultative Expert System for Finite Element Modelling of Strip Drawing," presented at the XIII North American Manufacturing Research Conference, University of California at Berkeley, (May 1985).

[26] H. Furuta, K.S. Tu and J.T.P. Yao, "Structural Engineering Applications of Expert Systems," Computer Aided Design, vol. 17, 410-420 (1985).

[27] I.C. Taig, "Expert Aids to Finite Element System Applications," Proceedings of the 1st. International Conference, Southampton University, U.K.(edited by D.Sriram and R. Adey), 759-770 (1986).

[28] R.H. Holt and U.V.L. Narayana, "Adding Intelligence to Finite Element Modelling," proceedings of the Expert Systems in Government Symposium, (edited by K.N. Karna, K. Parasaye and B.G. Silverman), pp. 326-337, IEEE, New York (1986).

[29] A. Kissil and H.A. Kamel, "An Expert System Finite Element Modeler," Proceedings of the 1st. International Conference, Southampton University, U.K.(edited by D.Sriram and R. Adey), 1179-1186 (1986).

[30] B.W.R. Forde and S.F. Stiemer, "ESA: Expert Structural Analysis for Engineers," technical note in Computers and Structures, Vol. 29, 171-174 (1988).

[31] E.G. Schlechtendahl, "Specification of a CAD*I Neutral File for CAD Geometry" Version 3.2, Springer-Verlag, 1987.