WL-TR-96-3005



ASTROS ENHANCEMENTS

Volume II — ASTROS PROGRAMMER'S MANUAL

D.J. NEILL
D.L. HERENDEEN
R.L. HOESLY

Universal Analytics, Inc.
3625 Del Amo Blvd. Suite 370
Torrance, CA 90503

May 1995

FINAL REPORT FOR THE PERIOD JANUARY 1987 - APRIL 1995

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED

FLIGHT DYNAMICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-7562

DTIC QUALITY INSPECTED 1

19960514 014

NOTICE

WHEN GOVERNMENT DRAWINGS, SPECIFICATIONS, OR OTHER DATA ARE USED FOR ANY PURPOSE OTHER THAN IN CONNECTION WITH A DEFINITE GOVERNMENT-RELATED PROCUREMENT, THE UNITED STATES GOVERNMENT INCURS NO RESPONSIBILITY OR ANY OBLIGATION WHATSOEVER. THE FACT THAT THE GOVERNMENT MAY HAVE FORMULATED OR IN ANY WAY SUPPLIED THE SAID DRAWINGS, SPECIFICATION, OR OTHER DATA. IS NOT TO BE REGARDED BY IMPLICATION, OR OTHERWISE IN ANY MANNER CONSTRUED, AS LICENSING THE HOLDER, OR ANY OTHER PERSON OR CORPORATION; OR AS CONVEYING ANY RIGHTS OR PERMISSION TO MANUFACTURE, USE, OR SELL ANY PATENTED INVENTION THAT MAY IN ANY WAY BE RELATED THERETO.

THIS REPORT IS RELEASABLE TO THE NATIONAL TECHNICAL INFORMATION SERVICE (NTIS). AT NTIS, IT WILL BE AVAILABLE TO TEH GENERAL PUBLIC, INCLUDING FOREIGN NATIONS.

THIS TECHNICAL REPORT HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION.

Raymond M. Kolonay
Project Engineer
Design & Methods Development Section

Nelson D. Wolf, Technical Manager
Design & Methods Development Section
Design Development Branch

George R. Holderby, Chief
Design Development Branch
Structures Division

IF YOUR ADDRESS HAS CHANGED, IF YOU WISH TO BE REMOVED FROM OUR MAILING LIST, OR IF THE ADDRESSEE IS NO LONGER EMPLOYED BY YOUR ORGANIZATION, PLEASE NOTIFY WL/FIBAD BLDG 45, 2130 EIGHTH STREET, SUITE 1, WRIGHT-PATTERSON AFB OH 45433-7552 TO HELP MAINTAIN A CURRENT MAILING LIST.

Copies of this report should not be returned unless return is required by security consideration, contractual obligations, or notice on a specified document.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	MAY 1995	FINAL 01/15/87--04/30/95

4. TITLE AND SUBTITLE
ASTROS ENHANCEMENTS
VOLUME II - ASTROS PROGRAMMER'S MANUAL

5. FUNDING NUMBERS
CONTR: F33615-87-C-3216
PE: 62201F
PROJ: 2401
TASK: 02
WORK UNIT: 82

6. AUTHOR(S)
D. J. NEILL
D. L. HERENDEEN
R. L. HOESLY

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
UNIVERSAL ANALYTICS INC.
3625 DEL AMO BLVD SUITE 370
TORRANCE CA 90503

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
FLIGHT DYNAMICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT PATTERSON AFB OH 45433-7562

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
WL-TR-96-3005

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

ASTROS(Automated STRuctural Optimization System) is a computer program for the multidisicplinary design and analysis of aerospace structures. ASTROS combines mathematical optimization algorithms with traditional structural analysis disciplines such as static forces, normal modes, static aeroelasticity, and dynamic aeroelasticity (flutter), all in a finite element context, to perform automated preliminary design of an aircraft structure. This report is a complete user's manual that documents the many features of ASTROS through version 12 of the software package. It also provides information on system architecture and other topics of interest. This report is Volume 3 of a set; Volume 1 (Wl-TR-93-3025) is the user's manual.

14. SUBJECT TERMS	15. NUMBER OF PAGES
ASTROS, OPTIMIZATION, AEROELASTICITY, FLUTTER FINITE ELEMENT METHOD, STRUCTURAL DESIGN, AEROSPACE STRUCTURES	794
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

TABLE OF CONTENTS

1. INTRODUCTION 1

2. ASTROS SOFTWARE DESCRIPTION 5

2.1. THE ASTROS SYSTEM 6

2.1.1. SYSGEN Components 6
2.1.2. ASTROS Components 8

2.2. MAJOR FUNCTIONAL CODE BLOCKS 9

2.3. CODE COMMON TO ASTROS AND SYSGEN 12

3. SYSTEM INSTALLATION 13

3.1. MACHINE DEPENDENT CODE 14

3.1.1. General Dependent Code 15
3.1.2. Database Dependent Code 34

3.2. THE SYSTEM GENERATION PROGRAM 58

3.2.1. Functional Module Definition 59
3.2.2. Standard Solution Algorithm Definition 63
3.2.3. Bulk Data Template Definition 63
3.2.4. Relational Schema Definition 67
3.2.5. Error Message Text Definition 68

3.3. GENERATION OF THE ASTROS SYSTEM 70

4. EXECUTIVE SYSTEM 71

5. ENGINEERING APPLICATION MODULES 83

6. APPLICATION UTILITY MODULES 275

7. LARGE MATRIX UTILITY MODULES 323

8. THE CADDB APPLICATION INTERFACE 343

8.1. CADDB BASIC DESIGN CONCEPTS 347

8.1.1. Physical Structure 348
8.1.2. Improvements Over Other Databases 348
8.1.3. Memory Requirements 349

8.2. THE GENERAL UTILITIES 350

8.3. THE DYNAMIC MEMORY MANAGER UTILITIES 363

8.4. UTILITIES FOR MATRIX ENTITIES 375

8.4.1. Creating a Matrix. 375
8.4.2. Packing and Unpacking a Matrix by Columns. 376
8.4.3. Obtaining Matrix Column Statistics. 376
8.4.4. Packing and Unpacking a Matrix by Terms. 377
8.4.5. Packing and Unpacking a Matrix by Strings. 378
8.4.6. Matrix Positioning. 379
8.4.7. Missing Matrix Columns. 380
8.4.8. Repacking a Matrix. 381

8.5. UTILITIES FOR RELATIONAL ENTITIES 398

8.5.1. Examples of Relational Entity Utilities. 398
8.5.2. Creating a Relation. 399
8.5.3. Loading Relational Data. 399
8.5.4. Accessing a Relation 400
8.5.5. Updating a Relational entry. 401
8.5.6. Other Operations. 402

8.6. UTILITIES FOR UNSTRUCTURED ENTITIES 424

8.6.1. Generating an Unstructured Entity 424
8.6.2. Accessing an Unstructured Entity. 425
8.6.3. Modifying an Unstructured Entity. 426

9. DATABASE ENTITY DESCRIPTIONS 435

10. APPLICATION PROGRAMMER NOTES 789

10.1. SOFTWARE STANDARDS 790

10.2. DYNAMIC MEMORY MANAGEMENT 792

10.3. CADDB APPLICATION INTERFACE 793

10.4. CODING PRACTICE — THE ASTROS INTERFACE 794

ALPHABETICAL INDEX OF SOFTWARE MODULES

ABOUND 84
ACTCON 86
AEROEFFS 88
AEROSENS 91
AMP 94
ANALINIT 97
APFLUSH 98
APPEND 276
AROSNSDR 99
AROSNSMR 103
ASTROS 72
BCBGPDT 105
BCBULK 106
BCEVAL 107
BCIDVAL 109
BCIDVL 109
BKLEVA 234
BKSENS 235
BLASTDRV 110
BLASTFIT 112
BLASTRIM 114
BOUND 115
BOUNDUPD 117
CDCOMP 324
CEIG 325
COLMERGE 327
COLPART 328
CONORDER 118
DAXB 277
DBCINI 77
DBCLOS 352
DBCREA 353
DBDEST 354
DBEQUV 355
DBEXIS 356
DBFLSH 357
DBINIT 77
DBMDAB 35
DBMDAN 36
DBMDC1 38
DBMDC2 38
DBMDCH 37
DBMDDT 40
DBMDER 41
DBMDFP 42
DBMDHC 43
DBMDHX 44
DBMDI1 45
DBMDI2 46
DBMDLC 47
DBMDLF 48
DBMDMM 49
DBMDOF 50
DBMDOR 51
DBMDRD 52
DBMDSI 54
DBMDTR 55
DBMDWR 56

DBMDZB 57
DBNEMP 358
DBOPEN 359
DBRENA 361
DBSWCH 362
DBTERM 82
DCEVAL 119
DDLOAD 120
DECOMP 329
DESIGN 122
DESPUNCH 124
DMA 125
DOUBLE 16
DYNLOAD 127
DYNRSP 129
EBKLEVAL 131
EBKLSENS 132
EDR 133
EMA1 136
EMA2 138
EMG 140
FBS 330
FCEVAL 143
FLUTDMA 144
FLUTDRV 146
FLUTQHHL 147
FLUTSENS 150
FLUTTRAN 153
FNEVAL 155
FPKEVL 156
FREDUCE 157
FREQSENS 160
FSD 162
GDR1 164
GDR2 166
GDR3 167
GDR4 169
GDVGRAD 171
GDVRESP 172
GFBS 331
GMMATC 278
GMMATD 279
GMMATS 280
GPSP 173
GPWG 174
GREDUCE 175
GTLOAD 177
IFP 179
INERTIA 181
INVERC 281
INVERD 282
INVERS 283
ITERINIT 182
LAMINCON 183
LAMINSNS 184
LODGEN 186
MAKDFU 188
MAKDFV 190
MAKDVU 192
MAKEST 193
MAPOL 79
MERGE 332
MK2GG 196
MKAMAT 197

MKDFDV 199
MKDFSV 200
MKPVECT 202
MKUSET 203
MMBASC 366
MMBASE 367
MMDUMP 368
MMFREE 369
MMFREG 370
MMGETB 371
MMINIT 76
MMREDU 372
MMSQUZ 373
MMSTAT 374
MPYAD 333
MSGDMP 284
MSWGRESP 205
MXADD 335
MXFORM 382
MXINIT 383
MXNPOS 384
MXPAK 385
MXPKT 386
MXPKTF 387
MXPKTI 388
MXPKTM 389
MXPOS 390
MXRPOS 391
MXSTAT 392
MXUNP 393
MXUPT 394
MXUPTF 395
MXUPTI 396
MXUPTM 397
NLEMA1 206
NLEMG 209
NLLODGEN 211
NREDUCE 213
OFPAEROM 215
OFPALOAD 217
OFPDISP 220
OFPDLOAD 223
OFPEDR 225
OFPGRAD 227
OFPLOAD 228
OFPMROOT 230
OFPSPCF 231
PARTN 336
PBKLEVAL 234
PBKLSENS 235
PFBULK 236
POLCOD 285
POLCOS 286
POLEVD 287
POLEVS 288
POLSLD 289
POLSLS 290
PREPAS 74
PS 291
PVCDRV 202
QHHLGEN 238
RBCHECK 240
RDDMAT 292
RDSMAT 293

REAB 403
READ 404
READD 405
READDM 406
RECLRC 407
RECOND 408
RECOVA 242
RECPOS 409
REENDC 410
REGB 411
REGBM 412
REGET 413
REGETM 414
REIG 337
RENULx 415
REPOS 416
REPROJ 417
REQURY 418
RESCHM 419
RESETC 420
RESORT 421
REUPD 422
REUPDM 423
ROWMERGE 338
ROWPART 339
SAERO 244
SAERODRV 248
SAEROMRG 250
SAXB 294
SCEVAL 252
SDCOMP 340
SHAPEGEN 295
SOLUTION 255
SPLINES 257
SPLINEU 259
STEADY 261
STEADYNP 263
TCEVAL 265
TRIMCHEK 267
TRNSPOSE 341
UNGET 427
UNGETP 428
UNPOS 429
UNPUT 430
UNPUTP 431
UNRPOS 432
UNSTAT 433
UNSTEADY 269
USETPRT 296
UTCOPY 297
UTCSRT 298
UTEXIT 299
UTGPRT 300
UTMCOR 301
UTMINT 302
UTMPRG,UTRPRG,UTUPRG 303
UTMPRT 304
UTMWRT 305
UTPAGE, UTPAG2 307
UTRPRT 308
UTRSRT 309
UTSFLG, UTSFLR, UTGFLG, UTGFLR 310
UTSORT 311
UTSRCH 312

UTSRT3 313
UTSRTD 314
UTSRTI 315
UTSRTR 316
UTSTOD, UTDTOS 317
UTUPRT 318
UTZERD 319
UTZERS 320
WOBJGRAD 271
XISTOI 321
XISTOR 322
XQENDS 81
XQINIT 73
XQTMON 80
XXBCLR 17
XXBD 18
XXBSET 19
XXBTST 20
XXCLOK 21
XXCPU 22
XXDATE 23
XXFLSH 24
XXINIT 25
XXITOS 26
XXLSFT 27
XXNOT 28
XXOVFL 29
XXRAND 30
XXRSFT 31
XXRTOS 32
XXULNS 33
YSMERGE 272

ALPHABETICAL INDEX OF DATABASE ENTITIES

AA 436
AAICMAT 436
ACPT 437
AECOMPS 440
AECOMPU 441
AEFACT 442
AERO 442
AEROGEOM 443
AEROS 444
AERUGEOM 444
AESURF 445
AF 446
AG 446
AGA 446
AICMAT 447
AICS 447
AIRFOIL 448
AIRFRC 449
AJJTL 449
AL 449
AMAT 450
AR 450
ASET 450
ASET1 451
ATTACH 451
AXSTA 452
BDD 452
BEAMEST 453
BFRC 456
BGPDT 456
BHH 457
BLAST 458
BLGTJA 459
BLSTJA 459
BODY 459
BTEM 460
CAERO1 461
CAERO2 462
CAERO6 463
CAROGEOM 464
CAROUGEO 465
CASE 466
CASELIST 470
CBAR 471
CELAS1 472
CELAS2 473
CENTRESP 473
CIHEX1 474
CIHEX2 475
CIHEX3 476
CLAMBDA 477
CMASS1 478
CMASS2 479
CONEFFF 479
CONEFFS 480
CONLINK 480
CONM1 481
CONM1EST 482

CONM2 483
CONM2EST 484
CONROD 485
CONST 486
CONSTORD 492
CONVERT 493
CORDRESP 493
CORD1C 494
CORD2C 494
CORD1R 495
CORD2R 495
CORD1S 496
CORD2S 496
CQDMEM1 497
CQUAD4 498
CROD 499
CSHEAR 499
CSTM 500
CTRIA3 501
CTRMEM 502
D 502
DCENT 503
DCONALE 503
DCONBK 504
DCONBKE 504
DCONCLA 505
DCONDSP 505
DCONEP 506
DCONEPM 506
DCONEPP 507
DCONF 507
DCONFLT 508
DCONFRQ 509
DCONFT 509
DCONFTM 510
DCONFTP 510
DCONLAM 511
DCONLIST 511
DCONLMN 512
DCONPMN 512
DCONSCF 513
DCONSDE 513
DCONSDL 514
DCONTH2 514
DCONTH3 515
DCONTHK 515
DCONTRM 516
DCONTW 516
DCONTWM 517
DCONTWP 518
DCONVM 519
DCONVMM 519
DCONVMP 520
DDELDV 520
DDMVI 521
DDPGRV 521
DDPTHV 522
DDVCT 523
DDVSIZE 524
DELB 525
DELM 525
DELTA 525
DENSLIST 525
DESELM 526

DESHIST 527
DESLINK 528
DESVARP 529
DESVARS 530
DFDU 531
DFDUF 531
DFDU 531
DFSV 532
DGMMCT 533
DISPDRVA 534
DISPRESP 534
DKUG 535
DKELM 535
DKVI 536
DKVI0 536
DKVIG 537
DK1V 537
DLAGS 538
DLOAD 538
DLONLY 539
DMAG 539
DMELM 540
DMIG 541
DMU 541
DMUA 541
DMUF 542
DMUG 542
DMUL 542
DMUN 542
DMUO 543
DMUR 543
DMVI 543
DMVI0 543
DMVID 544
DMVIG 544
DPAV 544
DPFV 544
DPGRVD 545
DPGRVI 545
DPGV 546
DPLV3 546
DPNV 546
DPOV 546
DPRV 547
DPTHVD 547
DPTHVI 547
DPVJ 548
DP1 548
DRHS 549
DTELM 549
DTSLP 550
DUAD 550
DUAV 550
DUFV 551
DUG 551
DULD 551
DULV 552
DURD 552
DVCT 553
DVCTD 555
DVSIZE 556
DVSIZED 557
DVTOPTE 557
DVTOPTL 558

DVTOPTP 558
DDWGH2 558
DWGH1 559
DWNWSH 559
DYNRED 559
D1JK 560
D2JK 561
EIDTYPE 561
EIGC 562
EIGR 563
ELAS 563
ELASEST 564
ELEMDRVA 564
ELEMLIST 565
ELEMRESP 565
ELIST 566
ELISTM 567
EOBAR 568
EODISC 571
EOELAS 572
EOHEX1 573
EOHEX2 576
EOHEX3 579
EOQDMM1 582
EOQUAD4 584
EOROD 587
EOSHEAR 588
EOSUMMRY 590
EOTRIA3 591
EOTRMEM 594
EPOINT 596
FARGLIST 596
FCHAIN 596
FDERLIST 597
FDAMPDVA 597
FDAMPRSP 598
FFREQDVA 598
FFREQRSP 599
FLCFDRVA 599
FLCFRESP 600
FFT 600
FLFACT 601
FLUTMODE 601
FLUTREL 602
FLUTTER 603
FNARO 604
FNCEN 605
FNCOR 605
FNDNV 606
FNELM 607
FNFLT 608
FNFRQ 609
FNGRD 610
FNMAS 611
FNTHK 611
FNWGH 612
FORCE 612
FORCE1 613
FREQ 613
FREQ1 614
FREQ2 614
FREQL 615
FREQLIST 615
FREQDRVA 615

FREQRESP 616
FROOTDVA 616
FROOTRSP 617
FTF 617
FTOKLIST 618
GASUBO 618
GDVLIST 619
GDVDRVA 619
GDVRESP 619
GENEL 620
GENFA 620
GENF 621
GENK 621
GENM 621
GENQL 621
GENQ 622
GFE 622
GFR 622
GEOMSA 623
GEOMUA 624
GGO 625
GLBDES 625
GLBSIG 627
GMKCT 628
GMKCT0 629
GMKCTG 630
GMMCT 631
GMMCT0 632
GMMCTD 633
GMMCTG 634
GPFDATA 635
GPFELEM 636
GPST 637
GPWGGRID 638
GRADIENT 639
GRAV 639
GRID 640
GRIDLIST 640
GRIDTEMP 641
GSKF 641
GSTKF 641
GSTKG 641
GSTKN 642
GSUBO 642
GTKF 642
GTKG 642
GTKN 643
GUST 643
IARGLIST 643
IC 644
ICDATA 644
ICMATRIX 645
ID2 645
IFM 645
IFR 646
IHEX1EST 646
IHEX2EST 647
IHEX3EST 648
ITERLIST 649
JOB 650
JSET 651
JSET1 651
KAA 651
KAAA 652

KAFF 652
KALL 652
KALR 652
KAO 652
KARL 652
KARR 653
KDDF 653
KDDT 653
KEE 653
KELM 654
KELMD 655
KEQE 655
KFF 655
KFS 656
KGG 656
KHHF 656
KHHT 657
KLL 657
KLLINV 657
KLLL 657
KLLU 658
KLR 658
KL11 658
KNN 658
KOA 659
KOO 659
KOOINV 659
KOOL 660
KOOU 660
KSOO 660
KSS 660
KU11 661
K11 661
K1112 661
K12 662
K21 662
LAMBDA 663
LAMDAC 664
LDVLIST 664
LHS 665
LKQ 665
LOAD 665
LOCLVAR 666
LSOO 667
MAA 667
MAABAR 667
MACHLIST 667
MASDRVA 668
MASRESP 668
MASSEST 669
MATSS 669
MATTR 670
MAT1 670
MAT2 671
MAT8 672
MAT9 673
MDD 674
MELM 675
MELMD 676
MFF 676
MFORM 676
MGG 677
MHH 677
MII 677

MKAERO1 678
MKAERO2 679
MLL 679
MLR 679
MNN 679
MOA 679
MODELIST 680
MOMENT 680
MOMENT1 681
MOO 681
MPART 681
MPC 682
MPCADD 682
MPPARM 683
MRR 683
MRRBAR 683
MXDSPDVA 683
MXELMDVA 684
MXFCFDVA 684
MXFDPDVA 684
MXFFQDVA 684
MXFRQDVA 684
MXFRTDVA 685
MXGDVDVA 685
MXMASDVA 685
MXTHKDVA 685
MXTRMDVA 685
MXWGHDVA 686
NLGLBSIG 686
NLSMAT 687
OAGRDDSP 688
OAGRDLOD 689
OCEIGS 690
OCPARM 690
OEIGS 691
OEULBUCK 692
OGPWG 693
OGRIDDSP 694
OGRIDLOD 695
OLOCALDV 696
OMIT 698
OMIT1 698
OPNLBUCK 699
OTL 699
OPTIMIZE 700
PA 701
PAA 701
PAERO1 702
PAERO2 703
PAERO6 704
PAF 704
PAL 704
PAR 705
PARBAR 705
PARL 705
PBAR 706
PBAR1 707
PCAS 707
PCOMP 708
PCOMPS 709
PCOMP1 710
PCOMP2 711
PDF 711
PDLIST 712

PDT 713
PELAS 713
PF 713
PFGLOAD 713
PFOA 714
PG 714
PGA 714
PGMN 715
PHIA 715
PHIB 715
PHIE 715
PHIF 716
PHIG 716
PHIKH 716
PHIN 716
PHIO 716
PHIOK 717
PHIR 717
PIHEX 718
PLBAR 718
PLIST 719
PLISTM 720
PLOAD 721
PLOAD2 721
PLOAD4 722
PLYLIST 722
PMASS 723
PMAXT 723
PMINT 724
PN 724
PNSF 724
PO 724
POARO 725
PQDMEM1 725
PR 725
PRAS 726
PROD 726
PS 727
PSHEAR 727
PSHELL 728
PTGLOAD 729
PTRANS 729
PTRMEM 729
P1 730
P2 730
QDMM1EST 731
QEE 732
QHHL 733
QHJL 733
QJJL 733
QKJL 734
QKKL 734
QRE 734
QRR 735
QUAD4EST 736
RANDPS 738
RBAR 739
RBE1 739
RBE2 740
RBE3 740
RGCFRESP 741
RHS 741
RLOAD1 742
RLOAD2 742

RODEST 743
RROD 744
R21 744
R22 745
R31 745
R32 745
SAVE 746
SEQGP 746
SET1 747
SET2 747
SHAPE 748
SHAPEM 749
SHEAREST 750
SHPGEN 751
SKJ 751
SLPMOD 752
SMAT 752
SMPLOD 753
SPC 754
SPC1 754
SPCADD 755
SPLINE1 755
SPLINE2 756
SPOINT 756
STABCF 757
SUPORT 758
TABDMP1 759
TABLED1 759
TELM 760
TELMD 760
TEMP 761
TEMPD 761
TF 762
TFDATA 763
TFIXED 764
THKDRVA 764
THKRESP 765
TIMELIST 765
TLOAD1 766
TLOAD2 766
TMN 767
TMP1 767
TMP2 767
TREF 768
TREFD 768
TRIM 769
TRIMDATA 770
TRIMDRVA 771
TRIMRESP 771
TRIA3EST 772
TRMEMEST 775
TSTEP 777
UA 777
UBLASTG 778
UBLASTI 778
UDLOLY 778
UF 778
UFREQA 779
UFREQE 779
UFREQF 779
UFREQG 779
UFREQI 780
UFREQN 780
UG 780

UGA 781
UGTKAB 781
UGTKA 781
UGTKF 781
UGTKG 781
UGTKN 782
UGTKO 782
UKQ 782
UM 782
UN 782
UNMK 783
UO 783
UOO 784
URDB 784
USET 785
UTRANA 785
UTRANE 786
UTRANF 786
UTRANG 786
UTRANI 786
UTRANN 787
VELOLIST 787
VSDAMP 787
WGHDRVA 788
WGHRESP 788
YS 788

FOREWORD

This interim report is submitted in fulfillment of CDRL CLIN 0001, Sequence No. 12. of the ASTROS Enhancements Contract, F33615-87-C-3216, dated 29 January 1987. This volume provides the basic programmer's documentation for the Automated Structural Optimization System.

This work was performed by Universal Analytics, Inc. and their subcontractor, Northrop Corp. This manual supercedes the interim ASTROS Programmer's Manual documented in WL-TR-93-3038, Volume II by D.J. Neill, D.L. Herendeen and R.L. Hoesly. The major contributors to this report were D.L Herendeen, the Program Manager, D.J. Neill, the Associate Program Manager, and R.L. Hoesly, the chief programmer for the effort.

Capt. R. Canfield, Capt. S. Pitrof and Mr. R. Kolonay have been the Air Force Project Engineers and Dr. V.B. Venkayya was the initiator of the effort. This report covers the work performed between 29 January 1987 and 31 March 1995.

1. INTRODUCTION

This Programmer's Manual is one of the manuals documenting the Automated STRuctural Optimization System (ASTROS). The other three are the Theoretical Manual, the User's Manual and the Applications Manual. The Theoretical and Applications Manual indicate the range of capabilities of the ASTROS system while the User's Manual provides a complete description of the user interface to the system. The Programmer's Manual gives the detailed description of ASTROS software. It describes the system in terms of its software components, documents the procedure for installing ASTROS on different host machines and provides detailed documentation of the application and utility modules that comprise the procedure. In addition, the data structures of the database entities are presented in detail. This manual is intended to provide the system administrator with a guide to the existing software and the researcher with sufficient information to add application modules or otherwise manipulate the data generated by the ASTROS system. Using standard ASTROS features does not require a familiarity with the information contained in this manual except, perhaps, for the entity documentation, which is useful when additional database entities are to be viewed.

This document, while useful to the advanced engineering user, is directed toward the system administrator or code developer. This is the individual referred to by the term *user* unless otherwise indicated. The Programmer's Manual is structured in this way because all the information needed by the engineering user is contained as a subset of that needed by the system administrator. As a consequence, however, the manual is not as simple for the analyst as might be desired. It is anticipated that the advanced application user will need to sift through the module documentation and entity documentation to extract the information needed to modify the ASTROS execution path or to insert additional modules for performing alternative computations, printing additional results, writing data in alternative formats or other advanced features that may be performed.

As an introduction to the ASTROS system, Section 2 contains a description of the software structure of ASTROS, both to provide a resource for the system administrator and to be a road map for the application user in identifying specific modules relevant to the task of interest. Section 2 attempts to introduce the user to the totality of ASTROS source code and their interrelationships so that subsequent reading will be more readily interpretable: in essence, Section 2 provides a nomenclature section enabling the reader to identify (with the inevitable exceptions) the major unit (module) or functional library to which a particular program belongs. This section provides a framework for subsequent sections in the Programmer's Manual.

Section 3 is devoted to the installation of the ASTROS system on various host computers. The steps involved in installing the system are given, followed by detailed documentation of all the machine and installation dependent code. Sufficient detail is given to allow someone familiar with the target host system to write a set of machine-dependent code for that machine or site. This documentation is followed by the description of the System Generation Process (SYSGEN) and its inputs. These inputs, along with the SYSGEN program, define the system database which, in turn, defines system data to the ASTROS executive. It is these inputs which the researcher may wish to modify to define a new module, define a new set of inputs or make other advanced modifications of the system. A brief presentation of the order of the operations that follow preparation of the machine dependent library is given to complete an installation of the system.

Sections 4 through 8 contain the formal documentation of the ASTROS modules. Section 4 documents those portions of the code that are considered to be at the system level. This means that the user need not be aware of their existence but they are important in the overall system architecture. Further, they perform many tasks of which the user may want to be aware if any system modifications are to be made. Sections 6 through 8 document the utilities that are associated with the ASTROS application modules, matrix operations and the database. These sections are the most important from the view of the advanced researcher/user in that these are the software tools from which additional capabilities can be put together with reasonable rapidity. In each case, the executive (MAPOL) and application interface is fully defined and the algorithm of the utility is outlined.

Section 9 contains the documentation of the data structures on the CADDB database that are used by the ASTROS system. The contents and structure of each database entity are given along with an indication of the module that generates the data and which modules use the data. For matrix entities, the relevant section of the Theoretical Manual is also referenced since the entity contents are more clearly understood in the content of the equations that are highlighted there.

Section 10 contains a presentation of notes for the ASTROS application programmer. It is felt that the ASTROS system has been designed with sufficient flexibility that the additional features or minor enhancements are desired. Section 10, therefore, attempts to address some issues involved in writing an ASTROS module. Rules and guidelines are given which will help the programmer avoid complications arising from the interface of the new module and application utilities are also given. Particular emphasis is placed on the memory management utilities and the database utilities since these require a more sophisticated interface than the simple application utilities.

A standard documentation format has been adopted for the modules that are described in Section 3 through 8. Figure 1 illustrates this format and provides a key for identifying the data that are given for each module. While this format is brief, enough information is given for the user to identify the principal action of the module and the role it plays in the standard ASTROS execution. The utility modules are documented to the extent necessary for an application programmer to use the utility in any new code to be inserted in the system.

< module type> Module: **<name>**

Entry Point: **<FORTRAN calling list for module>**

PURPOSE:

<one or two sentence description>

MAPOL Calling Sequence:

<Executive system access method>

Application Calling Sequence:

<FORTRAN call followed by input description>

Method:

<Description of the module's action>

Design Requirements:

<Indicates what the module expects to have completed in the context of the standard sequence>

Error Conditions:

<Brief description of some error conditions that are trapped by the module>

Figure 1. Module Documentation Format

THIS PAGE LEFT INTENTIONALLY BLANK

2. ASTROS SOFTWARE DESCRIPTION

ASTROS is a software system made up of two separate executable programs comprising over 1500 independently addressable code segments containing approximately 300,000 lines of FORTRAN. While this Programmer's manual is devoted primarily to the detailed documentation to the separate modules and subroutines of the ASTROS system, an overview of that system is necessary to understand how the individual pieces fit together. This section introduces the ASTROS system and describes the software structure of ASTROS in terms of its major code blocks. Both the system generation program, SYSGEN, and the main program, ASTROS, are described and their interrelationships are illustrated. This section provides a resource for the system administrator and a road map for the application programmer to identify the section documenting modules relevant to the task of interest. This section also provides a framework to direct the subsequent sections in the Programmer's Manual.

In the context of the Programmer's Manual, the structure of the ASTROS system refers to the interrelationships among the major code blocks. Typically, an analysis of the software associated with an individual code segment will indicate the nature of the task being performed and provide information on the mechanisms by which intramodular communication takes place. The larger picture, in which the intermodular requirements of a particular code segment becomes clear, is more difficult to grasp. It is that picture which this section attempts to provide.

The magnitude of the ASTROS system requires that the code segments be grouped into abstract collections of code such as utility modules and the database in order to be understood. While necessary, these abstract collections can also obscure the picture of the system since a great deal of the detail is necessarily lost. Nonetheless, since a discussion of each individual code segment is not possible, a set of code blocks has been defined for the purpose of writing the Programmer's Manual. Naturally, there are many ways in which the code segments could be grouped to aid the user in understanding the code segments and their interactions. For the Programmer's Manual, the code is grouped in a hierarchical manner by function: that is, code segments that perform similar tasks at a similar level (relative to the executive system) are grouped together. Some segments of the code, of course, do not fit clearly into this sort of functional abstraction. Their role is such that they could lie in more than one group or really don't belong to any group that has been defined. These exceptions complicate the issue but do not destroy the utility of the functional breakdown of the code. When a module could be documented with more than one code group, this fact is noted in the appropriate manual sections.

2.1. THE ASTROS SYSTEM

The highest level of abstraction is illustrated in Figure 2, which presents the two executable images that comprise the ASTROS system, their inputs, outputs, and interrelationships. Referring to the figure, each of the illustrated components is briefly described in the following sections.

2.1.1. SYSGEN Components

The SYSGEN program is a stand-alone executable program that is used to define ASTROS system parameters. The use of an executable program that is directed by a set of inputs was adopted to provide a simple mechanism to expand the capabilities of the ASTROS procedure. The inputs, outputs, and use of this very important feature of the ASTROS architecture are fully documented in Section 3.2. The SYSGEN program consists of five items indicated by the numbered boxes in Figure 2. Each of these is briefly discussed below:

1. The SYSGEN INPUTS consist of a set of files that define certain system level data that is written by SYSGEN to the system database, SYSDB.

2. SYSGEN is an executable program that reads the SYSGEN INPUTS and creates a set of database entities on SYSDB that provide data to the ASTROS executive and high level engineering modules.

3. The SYSTEM DATABASE, SYSDB, consists of an index file, SYSDBIX, and (typically) a single data file, SYSDB01. The SYSGEN program creates and loads database entities onto the system database which defines:

   a. The set of modules which can be addressed through the MAPOL language

   b. The set of relational schemata for all relations declared in the MAPOL sequence

   c. The set of input Bulk Data entries

   d. The error message texts for most run time error messages

   e. The standard MAPOL sequence to direct the execution of the ASTROS

4. **XQDRIV** is a FORTRAN subroutine written by SYSGEN that must be compiled and linked into the ASTROS executable during the generation of the ASTROS executable image. It is the **XQDRIV** subroutine that forms the FORTRAN link between the MAPOL language and the application/utility modules.

5. The SYSGEN OUTPUT FILE is a listing generated by SYSGEN that echoes all the data stored on the system database. As such, it provides a resource for the application user and the system administrator documenting the current ASTROS system. Since this file represents what is, by definition, the ASTROS program, any problems that arise or questions in the documentation should be checked against the data in this file. If any discrepancies exist, either the documentation is in error or the SYSGEN inputs are in error. In any case, the ASTROS program is directed by the SYSGEN data.

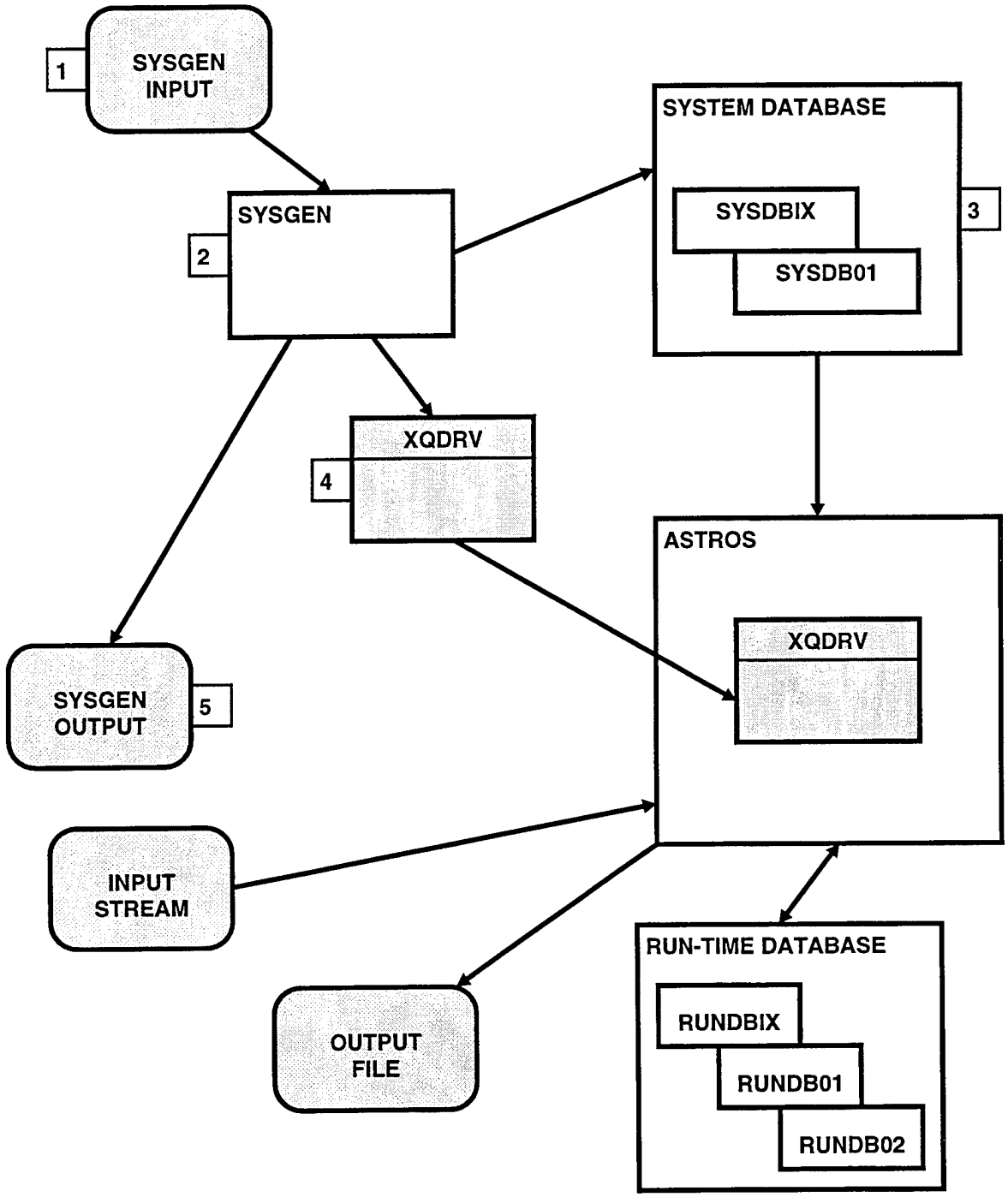


Figure 2. ASTROS System Overview

2.1.2. ASTROS Components

As illustrated in Figure 2, the **XQDRIV** subroutine and SYSDB are also part of the ASTROS program. The **XQDRIV** subroutine is needed to generate the executable image and the SYSDB files MUST be available on a read-only basis by the ASTROS program whenever an ASTROS job is run. The ASTROS program is comprised of the following:

1. **XQDRIV** is a FORTRAN subroutine written by SYSGEN that must be compiled and linked into the ASTROS executable during the generation of the ASTROS executable image. It is the **XQDRIV** subroutine that forms the FORTRAN link between the MAPOL language and the application/utility modules.

2. The SYSTEM DATABASE, SYSDB, contains database entities which define sets of data establishing the extent of some of the capabilities of the ASTROS program. ASTROS requires these files on a read-only basis for every execution of the system.

3. The ASTROS program is the main executable image associated with the ASTROS procedure. It is comprised of all the executive, database, utility, and engineering application modules that are needed to perform the automated multidisciplinary optimization tasks.

4. The INPUT STREAM is the user's input file containing the directives to execute the ASTROS program. The User's Manual is devoted to its documentation.

5. The OUTPUT FILE contains the data written to the user's output file containing those results of the ASTROS execution that were requested to be printed or that are printed by default.

6. The RUN-TIME DATABASE consists of one index file and one or more data files (called, respectively, RUNDBIX, and RUNDB01, 02, etc., in Figure 2) that contain the database generated at run time by ASTROS. Assuming an execution based on the standard MAPOL sequence, the run-time database will contain some or all of the entities that are documented in Section 9 of this manual. The application user can direct whether this database is to be saved or deleted on termination of the execution. The Interactive CADDB Environment (ICE) (AFWAL-TR-88-3060, August 1988) can be used to view these data, prepare reports or port the data into other applications.

2.2. MAJOR FUNCTIONAL CODE BLOCKS

Figure 3 presents a grouping of source code blocks within the ASTROS system. This grouping is functional in that code related to the performance of one task or a series of tasks at the same level relative to the executive system are grouped together. According to this breakdown, there are seven major blocks of code within ASTROS executable programs. The SYSGEN program has no executive system and is directed by a simple FORTRAN driver called SYSGEN. The ASTROS system, on the other hand, has a highly developed executive system that comprises this major ASTROS code block. Also shown are the five groups of routines which are used by the SYSGEN and ASTROS programs.

The naming conventions used within each code block are worthy of some discussion since they are useful in identifying an unknown routine in a piece of ASTROS software. Whenever possible, a set of consistent, meaningful mnemonics was adopted to identify groups of code that belong together, either functionally or logically. Where such conventions have been adopted, they are indicated in the discussion of the code block. One complication to such conventions is the use of existing source code as a resource for the ASTROS program. When major code units were used from existing software, the convention was not typically enforced. As a result, there are exceptions to the nomenclatures adopted in some of the source code blocks presented in this section.

Each of the source code blocks is now briefly discussed by reference to the name assigned to it in Figure 3 and its related Programmer's Manual section is indicated.

1. SYSGEN is a very small code block containing the SYSGEN driver (SYSGEN), a set of four output routines (**xxxOUT**) to print the SYSGEN output file and five routine (**TIMxxx**) that compute the timing constants for the large matrix utilities. The SYSGEN program has a single execution path which is documented in Section 3.2.

2. The ASTROS executive is the code block containing the ASTROS main driver program, ASTROS, and the ASTROS executive system software. The executive system is embodied in the routines beginning with the mnemonics **XQxxxx**. In addition to the pure executive system routines, the executive initialization routines for the database (**DBINIT**) and the memory manager (**MMINIT**) are also located in this code block. Finally, the general initialization routine **PREPAS** and the MAPOL compiler software are considered, for the purposes of the Programmer's Manual, to be part of the executive system. These routines are documented in Section 5.

3. The DATABASE code block contains all the software related to the application interface to the database and memory management systems for the ASTROS procedure. This software is further subdivided into five groups of code that represent the application interface to the database and memory manager. These groups are:

   a. The General Utilities that comprise the database application interface applicable to all database entity types. These routines are denoted by the mnemonics **DBxxxx** and are documented in Section 8.2.

   b. The Memory Management Utilities that comprise the application interface to the ASTROS dynamic memory manager. These routines are denoted by the mnemonics **MMxxxx** and are documented in Section 8.3.

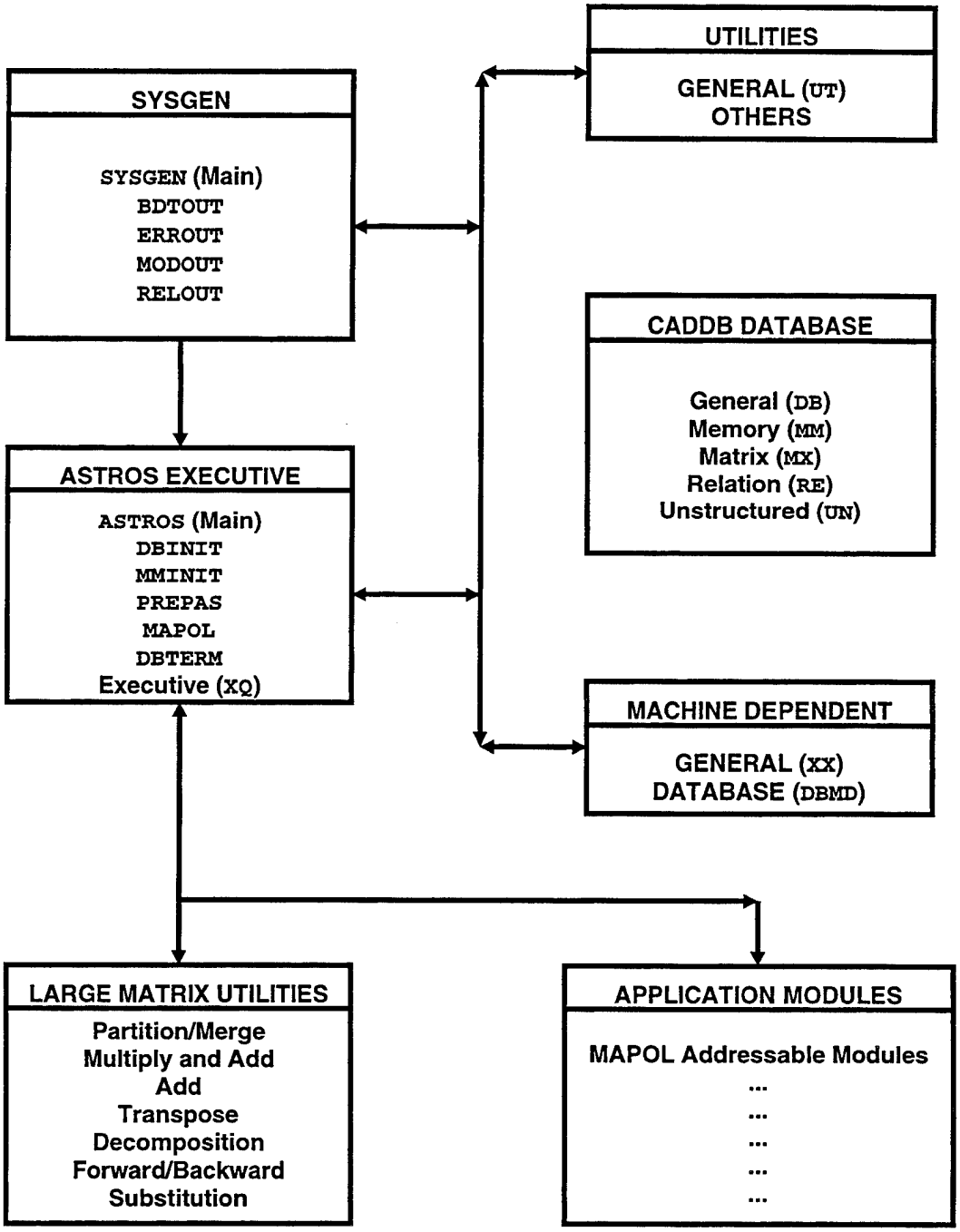


Figure 3. ASTROS Code Blocks

c. The Matrix Utilities that comprise the database application interface applicable to matrix entities. These routines are denoted by the mnemonics **MXxxxxx** and are documented in Section 8.4.

d. The Relation Utilities that comprise the database application interface applicable to relational entities. These routines are denoted by the mnemonics **RExxxxx** and are documented in Section 8.5.

e. The Unstructured Utilities that comprise the database application interface applicable to unstructured entities. These routines are denoted by the mnemonics **UNxxxxx** and are documented in Section 8.6.

4. The MACHINE DEPENDENT code block contains all the software in the ASTROS system that has been designated machine dependent. This software supplies the interface between the host computer and the ASTROS system. It is further subdivided into two groups of code:

   a. The General Utilities, comprising the machine dependent code used throughout the ASTROS system. These routines are denoted by the mnemonics **XXxxxxx** and are documented in Section 3.1.1.

   b. The Database Utilities, comprising the database machine dependent code used primarily by the database software. These routines are denoted by the mnemonics **DBMDxxx** and are documented in Section 3.1.2.

5. The UTILITIES code block contains all the machine independent application utilities developed for the ASTROS system. This software is a suite of functions that are useful in many places in the code. They have therefore been formalized to the extent that they may be used by any ASTROS application routine. The majority of these routines are denoted by the mnemonics **UTxxxxx** with exceptions corresponding to those in-core utilities that came from COSMIC/NASTRAN. These are documented in Section 6.

6. The LARGE MATRIX UTILITIES code block contains the utilities developed for the ASTROS system to operate on large matrices stored on the ASTROS database (rather than matrices stored in memory). This software comprises a suite of matrix operations that have been formalized to the extent that they may be used by any ASTROS application routine and by the ASTROS executive system. There is no consistent naming convention for these routines since they have been derived from their COSMIC/NASTRAN counterparts. The utilities are documented in Section 7.

7. The APPLICATION MODULES code block is the largest code block within ASTROS. It contains the engineering and application modules that support the analysis and optimization features of the ASTROS system. Each of these modules has been designed to be independent of the other application modules to the maximum extent possible. Typically, consistent naming conventions have been used for routines within each module. Because of the disparate code resources that were used in the development of ASTROS, however, no globally consistent naming convention was adopted. Section 5 documents each of the modules in the application library.

2.3. CODE COMMON TO ASTROS AND SYSGEN

Since some machines require or can take advantage of an explicit knowledge of which routines are needed to create an executable image, this section attempts to indicate which portions of the source code blocks (as grouped in Figure 3) are utilized within the SYSGEN program. With the exception of the SYSGEN code block, all the illustrated code blocks are used by the ASTROS program. The source code blocks that are needed, in whole, or in part, by SYSGEN are (1) the SYSGEN code, (2) the DATABASE code, (3) parts of the MACHINE DEPENDENT code, (4) some of the UTILITIES and (5) parts of the ASTROS EXECUTIVE.

Rather than write and maintain separate code blocks to perform similar functions, SYSGEN makes use of the suite of general utilities in the UTILITIES CODE BLOCK. The machine dependent code block is also shared between ASTROS and SYSGEN.

One of the tasks of SYSGEN is to compile and store the standard executive sequence (written in the MAPOL language) onto the system database. Therefore, the SYSGEN program makes use of the ASTROS EXECUTIVE code block to supply the MAPOL compiler. In addition, the SYSGEN driver must perform the executive functions to initialize the memory manager and the database. Therefore, the **MMINIT** and **DBINIT** routines from the ASTROS EXECUTIVE code block are also used by SYSGEN.

3. SYSTEM INSTALLATION

A software system of the magnitude of ASTROS requires a formal installation of the system on each host computer. For ASTROS, the installation process can be broken into three distinct phases. In the first phase, the ASTROS/host interface is defined and the proper machine dependent code is written to create that interface. The second phase involves the generation of the executable image of the SYSGEN program and its execution. Finally, the ASTROS executable image is generated using the outputs from the SYSGEN program. The purpose of this section is to document all the machine dependent code in a generic manner and to indicate which parameters and routines are most likely to be site dependent and which are truly machine dependent. In the typical case, the system manager at each facility will be given the machine dependent library for the host system that is to be used. For completeness, however, sufficient detail is presented to allow someone familiar with the host system to write a new set of machine dependent code.

Following the formal documentation of the machine dependent interface is a discussion of the SYSGEN program and its inputs. The SYSGEN program is important in that it provides the advanced analyst/user with a mechanism to add features to the system. It is also important for system installation in that part of its output is required before the executable image of the ASTROS procedure can be generated. Again, in the typical case the user will be given a proper set of SYSGEN outputs but the utility of SYSGEN in increasing the capabilities of the system makes its complete documentation very useful to the majority of ASTROS users. Finally, a brief section is included to present the total ASTROS installation in a step by step manner to give an overall view of the process.

The information presented in these sections serves as a guide to the installation of ASTROS on alternative host machines, but the nature of the machine dependencies make it impossible to anticipate all contingencies that may arise. The installation of the ASTROS procedure on a new host computer can therefore be a complex task despite the relatively small number of machine dependent routines.

3.1. MACHINE DEPENDENT CODE

The machine dependent interface has been designed such that approximately 40 routines are needed to complete the connection between ASTROS and the host system. The development of the machine dependent interfaces can be done in a straightforward manner on most machines with more complexity required for sophisticated interfaces or for alternative host architectures. The typical ASTROS user will not be willing to perform any but the most rudimentary duplication of the standard, supported installation dependent interface, although anyone familiar with the host computer system could accomplish the task. Installation at sites using machines that are much like the ones already supported is fairly simple, although even the installation of ASTROS on identical host machines can require some modification to the machine dependent code since some parameters and code are site dependent as well as machine dependent.

The machine dependent code is separated into two libraries: the general library, denoted by names starting with **XX**, and the database machine dependent library, denoted by names starting with **DBMD**. The general library consists of timing routines, bit manipulation routines, some character string manipulation routines, a random number generator and a **BLOCK DATA** subroutine containing a number of machine and installation dependent parameters. The timing routines and the random number generator are site dependent in that each facility typically has a library of such routines. The **BLOCK DATA** contains such parameters as the open core size, the definition of logical units, output paging parameters and other site dependent parameters. The remainder of the routines are very simple and typically do not vary substantially from site to site, although they are different between machines. In some cases, the **XX**-routines are written in standard FORTRAN and are in the machine dependent library only because some host systems provide special routines to perform these tasks.

The database machine dependent library (**DBMD**) is much more complex than the general machine dependent library. The complication arises because of the flexibility of the machine dependent interface and because of the nature of the interface. Unlike the **XX** library, the **DBMD** library deals with file structures and I/O to the host system and with memory management. These issues are highly machine dependent and are further complicated because the translation of machine independent parameters like file names to the actual host system file name may need to be very flexible depending on the nature of the local host system. The **ASSIGN DATABASE** entry in ASTROS allows the user to enter machine dependent parameters associated with the data base file attachment. A major task in writing the **DBMD** library is the definition of these parameters and the rules for their use: in general they are used to enable the user to modify the default file attributes. For example, block sizes; or their location on a physical device, such as disk volume. The flexibility inherent in the machine dependent interface can cause difficulties in writing the **DBMD** code, however, in that the code developer may find it hard to differentiate those aspects of the interface that are free to be redefined from those that are required by ASTROS. In the authors' experience, however, the task has proven to be tractable for all host systems used thus far by using the existing routines as a model. The reader should be under no illusion, however, that the task of writing the **DBMD** machine dependent library is simple.

The following sections document the **XX** and **DBMD** machine dependent libraries in a machine independent manner. Each routine that is essential to the ASTROS interface, its calling sequence and its design requirements is listed. It is very important to appreciate that the actual machine dependent interface may require additional routines that are not documented in these sections. The only routines

that are shown here are those that are referenced by the machine independent portions of ASTROS. By definition, it is these routines that constitute the machine dependent interface. It is often desirable and sometimes necessary for the machine dependent code to call other machine dependent routines. These internal interfaces are not documented in this report because of their high degree of dependence on particular host machines and/or site configurations. It is completely up to the discretion of the code developer to decide whether such routines are desirable and what tasks they should perform. In fact, there are no requirements of any kind for the machine dependent code ***except*** those imposed by the definition of the interface (calling sequence and design assumptions). It is that very flexibility that makes the machine dependent code generation difficult.

3.1.1. General Dependent Code

The following sections document each of the general machine dependent routines contained in the **xx** library. These routines tend to be highly site dependent as well as machine dependent, but are relatively straightforward to develop. Their functions are simple and do not deal with the major machine dependencies like I/O and word sizes.

Machine Dependent Utility Module: **DOUBLE**

Entry Point: **DOUBLE**

Purpose:

Machine dependent logical function to determine the machine precision as one of single or double precision.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
DOUBLE ( )
```

Method:

DOUBLE returns a **.TRUE.** if the machine precision is double or a **.FALSE.** if it is single. ASTROS then produces ***all*** matrix entities and ***assumes*** that ***all*** matrix entities are of the machine precision. Mixing single and double precision matrices is ***not*** supported by ASTROS code. **DOUBLE** should be used by ***all*** application modules that use matrix entities.

Design Requirements:

1. All matrix operations must be either single or double, not mixed.

Error Conditions:

None

Machine Dependent Utility Module: **XXBCLR**

Entry Point: **XXBCLR**

Purpose:

Machine dependent integer function to clear a bit in an array.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
XXBCLR ( ARRAY, BIT )
```

Method:

The bit manipulation routines all assume that the **BIT** identifier can vary from 1 to any positive integer. A consistent set of assumptions on the correspondence of **BIT** to a word/bit combination in **ARRAY** must be made for all bit routines.

Design Requirements:

1. For machine independent use, application program units should size **ARRAY** based on 32 or fewer bits per word.

Error Conditions:

None

Machine Dependent Utility Module: **XXBD**

Entry Point: **XXBD**

Purpose:

A block data subroutine to initialize machine or installation dependent parameters.

MAPOL Calling Sequence:

None

Application Calling Sequence:

None

Method:

The **XXBD** block data establishes the values of machine dependent constants. These parameters include any constant that may be needed for the machine dependent library as well as the following installation or machine dependent values required by the ASTROS machine independent routines:

(1) The size of the open core common block **/MEMORY/** in single precision words.

(2) System dependent precision terms for the large matrix utilities and memory management

(3) The parameters identifying the name and password of the ASTROS system database. These must correspond to those used in the SYSGEN program.

(4) The number of bytes and bits in a single precision word, the number of characters that will be stored in a hollerith word and the FORTRAN format statement to read or write one hollerith word.

(5) The set of "large" and "small" numbers for the machine, including a large real number, a small real number, the square root of a small real number and the largest integer value supported by the host system.

(6) The installation dependent number of lines per page and the maximum number of output lines that will be used by the ASTROS page utility, **UTPAGE**.

(7) The ASTROS and SYSGEN version and release identifiers.

(8) The installation dependent set of logical unit numbers identifying the read/write/punch units and the unit to be used for the include files, intermediate storage of the executive timing summary and the queued storage of the error messages.

(9) System dependent null values for relational entity attribute types

Design Requirements:

1. The logical units specified in the **XXBD** block data must not conflict with those identified in the database machine dependent block data **DBBD** for the database files.

Error Conditions:

None

Machine Dependent Utility Module: **XXBSET**

Entry Point: **XXBSET**

Purpose:

Machine dependent routine to set a bit in an array.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL XXBSET ( ARRAY, BIT )
```

Method:

The bit manipulation routines all assume that the **BIT** identifier can vary from 1 to any positive integer. A consistent set of assumptions on the correspondence of **BIT** to a word/bit combination in **ARRAY** must be made for all bit routines.

Design Requirements:

1. For machine independent use, application program units should size **ARRAY** based on 32 or fewer bits per word.

Error Conditions:

None

Machine Dependent Utility Module: **XXBTST**

Entry Point: **XXBTST**

Purpose:

Machine dependent logical function to test a bit in an array.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
XXBTST ( ARRAY, BIT )
```

Method:

The bit manipulation routines all assume that the **BIT** identifier can vary from 1 to any positive integer. A consistent set of assumptions on the correspondence of **BIT** to a word/bit combination in **ARRAY** must be made for all bit routines.

Design Requirements:

1. For machine independent use, application program units should size **ARRAY** based on 32 or fewer bits per word.

Error Conditions:

None

Machine Dependent Utility Module: **XXCLOK**

Entry Point: **XXCLOK**

Purpose:

Machine dependent routine to return the time of day as a character string and as a number of seconds past midnight.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL XXCLOK ( TIME, ISEC )
```

TIME	Character string containing the time of day as **HH:MM:SS** (Character, Output)
ISEC	Integer number of seconds since midnight. (Integer, Output)

Method:

None

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **XXCPU**

Entry Point: **XXCPU**

Purpose:

Machine dependent routine to return the elapsed CPU time in seconds.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL XXCPU ( CPU )
```

CPU	Number of seconds of CPU time used since the job started. (Real, Output)

Method:

On the first call to **XXCPU**, the utility must initialize the system CPU timer and return 0.0 elapsed seconds. On subsequent calls, the elapsed CPU time in seconds is returned.

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **XXDATE**

Entry Point: **XXDATE**

Purpose:

Machine dependent routine to return the date as a character string **MM/DD/YY**.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL XXDATE ( TODAY )
```

TODAY	Character string containing the date as **MM/DD/YY**. (Character, Output)

Method:

None

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **XXFLSH**

Entry Point: **XXFLSH**

Purpose:

Machine dependent routine to flush any data in the buffer for a given logical unit.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL XXFLSH ( LU )
```

LU	The logical unit number of the file whose buffer is to be flushed. (Integer, Input)

Method:

The **XXFLSH** routine will typically be a return. On machines that support the ability to flush the I/O buffer for a file, however, the **XXFLSH** routine should call that routine to flush the buffer to the file.

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **XXINIT**

Entry Point: **XXINIT**

Purpose:

Machine dependent routine to perform general machine dependent initialization tasks.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL XXINIT
```

Method:

The **XXINIT** routine is typically used to enter machine dependent parameters relating to error handling by the host machine, the initialization of the machine dependent parameters that must be done at run time on certain machines and performing any other machine or installation dependent actions that may be useful. The **XXINIT** routine is called by the ASTROS main driver as the first executable statement of the ASTROS.

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **XXITOS**

Entry Point: **XXITOS**

Purpose:

Machine dependent routine to return the character representation of an integer.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL XXITOS ( N, V )
```

N	Input integer
V	Output character string

Method:

This routine may be written in standard FORTRAN 77 using the internal file feature to write the integer onto the character string. It is often more efficient to crack the integer into its constituent digits. Some machines have local utilities that may be used.

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **XXLSFT**

Entry Point: **XXLSFT**

Purpose:

Machine dependent integer function to shift bits to the left in an integer word.

Function Arguments:

XXLSFT (INT, NBIT)

INT	Input integer
NBIT	Integer number of bits to shift left

Method:

The machine independent use of this function requires that **NBIT** be less than the smallest number of bits in a word for any target machine (typically 32).

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **XXNOT**

Entry Point: **XXNOT**

Purpose:

Machine dependent integer function that returns the complement of **INT**.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
XXNOT ( INT )
```

INT	Input integer

Method:

None

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **XXOVFL**

Entry Point: **XXOVFL**

Purpose:

Machine dependent routine to test for floating point overflow or underflow and return a flag denoting which has occurred.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL XXOVFL ( J )
```

J	Integer value returned based on the over/underflow condition: = **1** floating point overflow exists = **2** no error condition = **3** floating point underflow exists

Method:

In the case of this special routine, if the host system does not have an **XXOVFL** type of routine, it is necessary to return a **J=2** value for all calls to **XXOVFL**. In this case, the host system will be relied upon to indicate the occurrence of a floating point error.

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **XXRAND**

Entry Point: **XXRAND**

Purpose:

Machine dependent function that returns a random single precision number between 0.0 and 1.0.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
XXRAND ( )
```

Method:

Returns uniformly distributed random numbers.

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **XXRSFT**

Entry Point: **XXRSFT**

Purpose:

Machine dependent integer function to shift bits to the right in an integer word.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
XXRSFT ( INT, NBIT )
```

INT	Input integer
NBIT	Integer number of bits to shift right

Method:

The machine independent use of this function requires that **NBIT** be less than the smallest number of bits in a word for any target machine (typically 32).

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **XXRTOS**

Entry Point: **XXRTOS**

Purpose:

Machine dependent routine to return the character representation of a real number.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL XXRTOS ( REL, STR )
```

REL	Input real number
STR	Output character string

Method:

This routine may be written in standard FORTRAN 77 using the internal file feature to write the real onto the character string. It is often more efficient to crack the real into its constituent digits. Some machines have local utilities that may be used.

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **XXULNS**

Entry Point: **XXULNS**

Purpose:

Machine dependent routine to return the used length of a character string.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL XXULNS ( STR, ULEN )
```

STR	Character string (Character, Input)
ULEN	The position of the last nonblank character (the first character in the string is character 1). (Integer, Output)

Method:

The **XXULNS** routine may be written in standard FORTRAN 77 using the **LEN** function to return the total length and then looking backwards for the first nonblank character. Certain hosts may benefit from a machine dependent approach when byte operations are expensive.

Design Requirements:

None

Error Conditions:

None

3.1.2. Database Dependent Code

The following sections document each of the database machine dependent routines contained in the **DBMD** library. These routines tend to be site independent, but are highly machine dependent. Their development on a new host system can become quite complex depending on the desired sophistication of the interface. These routines deal with file structures I/O and memory management as well as certain CPU critical string manipulation functions.

Machine Dependent Utility Module: **DBMDAB**

Entry Point: **DBMDAB**

Purpose:

To abort the execution of ASTROS due to a database or memory management fatal error.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBMDAB     ( FLAG )
```

FLAG	An integer input denoting whether the executive termination utility **XQENDS** is to be called or not. = **0**, call **XQENDS**, otherwise stop

Method:

The **DBMDAB** routine is set up to avoid the recursion that can occur due to the termination actions taken by the **XQENDS** termination utility. Since the database and memory manager are calling for the abort, the **XQENDS** routine's attempts to close the database files often cause the **DBMDAB** routine to be called again. Hence, the flag argument is input to denote that the abort condition is such that any attempts to close the database will cause recursion.

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **DBMDAN**

Entry Point: **DBMDAN**

Purpose:

Machine dependent integer function that returns the logical **AND** of **INT1** and **INT2**.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
DBMDAN ( INT1, INT2 )
```

INT1	Input integer
INT2	Input integer

Method:

None

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **DBMDCH**

Entry Point: **DBMDCH**

Purpose:

To convert a character variable of arbitrary length into an integer array with four hollerith characters per word.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBMDCH ( CVAR, IVAR, LEN )
```

CVAR	An input character variable of arbitrary length
IVAR	The output integer array containing the hollerith equivalent of **CVAR**
LEN	An input integer denoting the number of characters to be placed in **IVAR**. **LEN** should always be a multiple of four, this routine pads with blanks as needed.

Method:

The **DBMDCH** routine is used extensively by the database routines to convert user supplied character variables into hollerith integers for subsequent processing. It is critical for performance that this routine be efficient. For implementation purposes, it must be assumed that the input character string can be of any length, but the output hollerith variable must always have four characters per word. Any extra bytes left unused are filled with blanks.

The only way standard FORTRAN provides to convert character data to hollerith data is with an incore file operation using the FORTRAN read. While this method works on all machines, it is typically very slow and causes severe performance penalties. This method can be avoided in most cases since compilers typically pass two arguments for every character variable with the virtual argument containing the character length. The virtual argument either follows the character argument directly or is passed at the end of the list of actual arguments. Knowing this, this routine can usually be written using all integer data thereby producing much faster code.

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **DBMDCX**

Entry Point: **DBMDC1**

Purpose:

To perform phase 1 of database configuration initialization.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBMDC1 ( NWORD )
```

NWORD Number of words required in **DBNT** (Integer, Output)

Method:

Phase 1 of the database configuration normally involves the determination of default values for the database. The values that can be changed are defined in the **/DBCONS/** common block. These values can be hard coded in this routine, hard coded in the **DBBD** block data routine or read from a configuration file.

The only required function of the routine is to return the number of words in the system dependent portion of the **DBNT**.

Design Requirements:

None

Error Conditions:

None

Entry Point: **DBMDC2**

Purpose:

To perform phase 2 of database configuration initialization.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBMDC2 ( DBNT )
```

DBNT Database name table (Integer, Input)

Method:

When phase 2 of the database configuration is performed, the **DBNT** table has been allocated and partially initialized. This routine must initialize the system dependent portion of the table. The location of this data can be found as follows.

```
DBENTSD          = Z (DBNT + DBNSD)
Z(DBENTSD + xx)  = machine dependent data
```

It is also the responsibility of this routine to make sure that all of the following variables in **/DBCONS/** have legal values.

DBDFIL	default number of data files
DBMFIL	maximum number of data files
DBDEFD	default data file block size
DBDEFI	default index file block size
DBMAXE	maximum number of **ENT** entries
DBMAXD	maximum number of **DBNT** entries
DBMAXN	maximum number of **NST** entries
DBALGN	required buffer alignment

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **DBMDDT**

Entry Point: **DBMDDT**

Purpose:

To return the time and date in hollerith formats.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBMDDT ( DATE, TIME )
```

DATE	Date in form **MM/DD/YY** (2 integer words, output)
TIME	Time in form **HH:MM:SS** (2 integer words, output)

Method:

This subroutine should return the current date and time in the appropriate locations. Each value returned should be stored in two integer words with four hollerith characters per word.

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **DBMDER**

Entry Point: **DBMDER**

Purpose:

To handle machine and installation dependent error conditions for the database and memory manager.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBMDER ( ERROR )
```

ERROR	A character argument containing an error message identifier.

Method:

The **DBMDER** routine is intended to be used in two ways. The first, denoted by a blank character string on input, is to activate any machine dependent error handling. This is the interface to the **DBMDER** routine from the machine independent library. For example, the **DBMDER** routine typically invokes the host dependent mechanism to obtain a traceback to assist in locating the source of an error. The second interface, using nonblank character strings on input, is intended for use by the machine dependent (**DBMD**) library. In this function, the **DBMDER** routine typically writes out error messages identifying the nature of the (machine dependent) error condition. This is useful for error checking the file naming conventions, host I/O limitations, and other host dependent user interfaces to the ASTROS system.

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **DBMDFP**

Entry Point: **DBMDFP**

Purpose:

An integer function to reorder the bytes in an integer word.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
DBMDFP ( INUM )
```

INUM	An integer word whose bytes are to be reordered

Method:

On certain machines (notably VAX), the bytes in an integer word are stored in an order right to left. When hollerith data are used, this feature complicates the comparison of two hollerith words. This routine is called to reorder the bytes in an integer word to be left to right, independent of the storage format on the machine. On machines that do not swap bytes, the **DBMDFP** function value should be set equal to the **INUM** value.

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **DBMDHC**

Entry Point: **DBMDHC**

Purpose:

To convert an integer array with four hollerith characters per word into a character variable of arbitrary length.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBMDHC ( IVAR, CVAR, LEN )
```

IVAR	The input integer array containing the hollerith characters.
CVAR	An output character variable containing the character representation of the hollerith **IVAR**.
LEN	An input integer denoting the number of characters in **IVAR** to convert and place in **CVAR**. **LEN** should always be a multiple of four, the routine truncates or pads with blanks as needed.

Method:

The **DBMDHC** routine is used extensively by the database routines to convert hollerith integer into character variables for subsequent processing. It is critical for performance that this routine be efficient. For implementation purposes it must be assumed that the output character string can be of any length and the input hollerith variable must have four characters per word as generated by **DBMDHC**.

The only way standard FORTRAN provides to convert hollerith data to character data is with an incore file operation using the FORTRAN write. While this method works on all machines, it is typically very slow and causes severe performance penalties. This method can be avoided in most cases since compilers typically pass two arguments for every character length. The virtual argument either follows the character argument directly or is passed at the end of the list of actual arguments. Knowing this, this routine can usually be written using all integer data which produces much faster code.

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **DBMDHX**

Entry Point: **DBMDHX**

Purpose:

To dump a portion of memory in a hexadecimal or octal format.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBMDHX ( ARRAY, LEN )
```

ARRAY	Input array containing data to be dumped
LEN	Number of single precision words to dump

Method:

If a database error occurs, portions of the in core control tables are dumped to help diagnose the problem. It is most often desirable to see this data in a combined hex/octal and character format. This routine usually dumps the desired data in the following form:

OFFH OFFD .....hex/octal data .....character data

OFFH - hex/octal offset of the data from **/MEMORY/**

OFFD - decimal offset of the data from **/MEMORY/**

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **DBMDIX**

Entry Point: **DBMDI1**

Purpose:

Phase 1 of database I/O initialization.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBMDI1 ( NAME, STAT, RW, USRPRM, NFILE, NWORD )
```

NAME	Database name (Character, Input)
STAT	Database status (**OLD**, **NEW**, **TEMP**, **SAVE**, or **PERM**) (Input)
RW	Read/write flag (**RO**, **WO**, **R/W**, Input)
USRPRM	User parameters (Character, Input)
NFILE	Number of data files in database (Integer, Output)
NWORD	Number of words required for each file in **DBDB** (Integer, Output)

Method:

Phase 1 of the I/O initialization is responsible for determining two values: the number of data files in the database and the number of system dependent words required for each file in the **DBDB**. The **DBINIT** call is provided with an argument called **USRPRM**. The contents of this character string are completely machine dependent and can be used to specify any special processing. Examples of these fields are provided in Section 1 of the User's Manual.

The most difficult function of this routine is to determine the number of data files for a database. The following ways could be used.

1. If the database has a status of **NEW** or **TEMP**, the number of data files is either the default or entered using the **USRPRM**.
2. If the database has a status of **OLD**, the number of data files can either be a hard coded value (usually 1) or can be determined by opening files with the appropriate names until an open fails.

For OLD databases this routine should also determine the index and data file block sizes. This can usually be done by one of the following two methods.

1. Inquire as to the physical attributes of the file to determine the block sizes.
2. Do a sequential read of the first block of the index file and extract the index and data file block sizes that are stored there.

Design Requirements:

None

Error Conditions:

None

Entry Point: **DBMDI2**

Purpose:

Phase 2 of database I/O initialization.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBMDI2 ( DBDB  )
```

DBDB Database Descriptor Block (Integer, Input)

Method:

When phase 2 of the database I/O initialization is performed, the **DBDB** table has been allocated and partially initialized. This routine must initialize the system dependent portion of the table. There are also several words in the machine independent portion of the **DBDB** that must be initialized for each index and data file. The following code shows how these words are located.

For the Index file:

```
DBDDBO = DBDB + DBDIFB
```

For the Data files:

```
DBDBO = DBDB + DBDDTA + (IFILE-1)*LENDDE
```

For all files:

```
DBDBSD = Z(DBDBO+DBDOSD)
Z(DBDB+DBDIBS) = Index file block size in words
Z(DBDB+DBDDBS) = Data file block size in words
Z(DBDB+DBDMDF) = Maximum number of data files
Z(DBDB+DBDONB) = Current number of blocks in the file
Z(DBDB+DBDOMB) = Maximum number of blocks allowed in the file
Z(DBDBSD+XX) = Machine dependent data
```

This routine will typically do any physical open or assign calls that are required to make all the index and data files for this database available for processing.

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **DBMDLC**

Entry Point: **DBMDLC**

Purpose:

An integer function to provide the memory manager with an address of a character memory location in particular precisions.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
DBMDLC ( BASE, PREC, STAT )
```

BASE	An input character array whose absolute address is desired.
PREC	An input integer denoting the desired precision of the address = 0 byte address = 1 word address = 2 double precision word address
STAT	An output integer value that is nonzero if any error conditions occurred.

Method:

The routine determines the absolute address of **BASE** and modifies the offset value to account for the precision of the desired address. For example, the VAX machine returns the byte address from the system utility **%LOCF**. To obtain the word address for the VAX, the byte address is divided by the number of bytes per word (four). A check is made to determine if the byte address is an even multiple of four and/or eight to check the single and double word alignment.

Design Requirements:

1. This routine is identical to **DBMDLF** except that the **BASE** array in this routine is character rather than integer.

Error Conditions:

1. On certain machines, there is a requirement that the memory addresses be aligned on single and/or double word boundaries. This routine should perform these checks and return the proper **STAT** value if the required alignments are not met.

Machine Dependent Utility Module: **DBMDLF**

Entry Point: **DBMDLF**

Purpose:

An integer function to provide the memory manager with an address of an memory location in particular precisions.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
DBMDLF ( BASE, PREC, STAT )
```

BASE	An input integer array whose absolute address is desired.
PREC	An input integer denoting the desired precision of the address = **0** byte address = **1** word address = **2** double precision word address
STAT	An output integer value that is nonzero if any error conditions occurred.

Method:

The routine determines the absolute address of **BASE** and modifies the offset value to account for the precision of the desired address. For example, the VAX machine returns the byte address from the system utility **%LOCF**. To obtain the word address for the VAX, the byte address is divided by the number of bytes per word (four). A check is made to determine if the byte address is an even multiple of four and/or eight to check the single and double word alignment.

Design Requirements:

1. This routine is identical to **DBMDLC** except that the **BASE** array in this routine is integer rather than character.

Error Conditions:

1. On certain machines, there is a requirement that the memory addresses be aligned on single and/or double word boundaries. This routine should perform these checks and return the proper **STAT** value if the required alignments are not met.

Machine Dependent Utility Module: **DBMDMM**

Entry Point: **DBMDMM**

Purpose:

Initializes machine dependent parameters for the memory manager.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBMDMM ( ICAWA, IWLIC )
```

ICAWA	An output integer indicating if characters are word aligned on the host machine: = 0 if character variables are word aligned = 1 if character variables are not word aligned
IWLIC	An output integer containing the number of characters stored in a single precision word on the host system.

Method:

None

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **DBMDOF**

Entry Point: **DBMDOF**

Purpose:

An integer function to return a FORTRAN index such that the location of one array can be accessed via another array

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
DBMDOF ( ARRAY1, ARRAY2 )
```

ARRAY1	One FORTRAN array (Input)
ARRAY2	Second FORTRAN array (Input)

Method:

The result, **DBMDOF**, is a FORTRAN index such that the same memory location is referenced by **ARRAY1** (**DBMDOF**) and **ARRAY2(1)**. In the **DBOPEN** call, the user provides a 20-word **INFO** array. The last 10 words of this block are available for any required user data. These 10 words can be modified anytime up to the **DBCLOS** call for the entity. Since the **INFO** array is not passed on the **DBCLOS** call, the **DBOPEN** call must remember where it is for later access by the **DBCLOS** call. The **DBMDOF** function allows the database to remember where the **INFO** block is by saving its location relative to the **/MEMORY/** common block at open time.

The actual implementation of the call usually requires some method for obtaining the actual address for a subroutine argument.

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **DBMDOR**

Entry Point: **DBMDOR**

Purpose:

Machine dependent integer function that returns the logical **OR** of **INT1** and **INT2**.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
DBMDOR ( INT1, INT2 )
```

INT1	Integer (Input)
INT2	Integer (Input)

Method:

None

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **DBMDRD**

Entry Point: **DBMDRD**

Purpose:

To read a block from the database.

MAPOL Calling Sequence:

None

Application Calling Sequence:

CALL DBMDRD (DBDB, FILE, BLK, BUFHD)

DBDB	Database Descriptor Block (Integer, input)
FILE	File Number (for index files, **FILE = 0**) (Integer, input)
BLK	Block Number; if **FILE < 0** then **BLK** is **IBLK*DBMFIL + FILE** (Integer, input)
BUFHD	The I/O header location (Intger, input)

Method:

The function of this routine is to read a block from the database. The first step is to determine the database file and block number to be read. the following code will perform this.

```
IF(FILE .LT. 0) THEN
     IBLK = BLK/DBMFIL
     IFILE = BLK - IBLK*DBMFIL
ELSE
     IBLK = BLK
     IFILE = FILE
ENDIF
```

The block should then be read into the I/O buffer using the appropriate calls for the target system. The machine independent **DBDB** data, referenced from **DBDBO**, and machine dependent **DBDB** data, referenced from **DBDBSD** can be obtained from the buffer header. The number of words to transfer, **BLKSIZ**, is obtained from the **DBDB**.

```
IF(IFILE .EQ. 0) THEN
     DBDBO = DBDB + DBDIFB
     BLKSIZ = Z(DBDB+DBDIBS)
ELSE
     DBDBO = DBDB + DBDDTA + (IFILE-1)*LENDDE
     BLKSIZ = Z(DBDB+DBDDBS)
ENDIF
BUFIO = Z(BUFHD+BFIOBF)
DBDBSD = Z(DBDBO+DBDOSD)
```

After the I/O operation, the following two words of the buffer header should be updated:

```
Z(BUFHD+BFPBLK) = IBLK*DBMFIL + IFILE
Z(BUFHD+BFDBDB) = DBDB
```

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **DBMDSI**

Entry Point: **DBMDSI**

Purpose:

To return the integer represented by a character string.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBMDSI ( STR, IVALUE )
```

STR	An input character string containing digits and signs representing an integer value.
IVALUE	An output integer variable containing the integer value represented by the input character string

Method:

This routine is typically written in standard FORTRAN, but may be available as a host system utility.

Design Requirements:

1. A leading + or – sign is permitted as are all the decimal digits. Any other characters are illegal.

Error Conditions:

1. If the character string does not represent an integer, no warnings are given and **IVALUE** is set to zero.

Machine Dependent Utility Module: **DBMDTR**

Entry Point: **DBMDTR**

Purpose:

To terminate processing of a database.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBMDTR ( DBDB )
```

DBDB	Database Descriptor Block (Integer, Input)

Method:

This routine is called at program termination to do any system dependent termination processing for each database. It is not required to do anything. Typically it will close all database files.

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **DBMDWR**

Entry Point: **DBMDWR**

Purpose:

To write a block to the database.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBMDWR ( BUFHD )
```

BUFHD I/O buffer header location (Integer, Input)

Method:

The function of this routine is to write a block to the database. The processing is similar to **DBMDRD** and the same information is available to the routine.

When writing this routine one special case must be considered. Because of the dynamic way in which database blocks are allocated and used, it can never be assumed that the database blocks are appended in sequential order. For example block 10 may be written before block 9. If this situation is not allowed on the target system then this routine should write dummy blocks to fill any gap before writing the target block. The contents of these dummy blocks is unimportant.

After the I/O operation, the "buffer modified" flag in the buffer header should be set to zero.

```
Z(BUFHD+BFMOD) = 0
```

Also, this routine should maintain the word in the machine independent portion of the **DBDB** which indicates the number of blocks on the physical file.

```
Z(DBDBO+DBDONB) = MAX(IBLK,Z(DBDBO+DBDONB))
```

Design Requirements:

None

Error Conditions:

None

Machine Dependent Utility Module: **DBMDZB**

Entry Point: **DBMDZB**

Purpose:

To find the first zero bit in a word

MAPOL Calling Sequence:

None

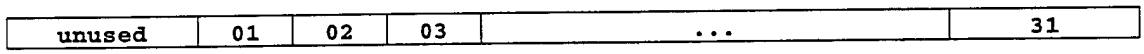
Application Calling Sequence:

```
CALL DBMDZB ( WORD, BITNO )
```

WORD	Word to be searched for a zero bit (Integer, Input)
BITNO	Bit number found. It will be a number ranging from 1 to 31 if a zero bit was found. It will be –1 if all 31 bits are on. (Integer, Output)

Method:

The free Block Bit Map (**FBBM**) uses a bit to represent each block on the particular database file. If the bit is on, the block is allocated and if the bit is off, the block is unallocated. Each word in the **FBBM** is used to represent 31 blocks. The bits are numbered as follows:



This bit numbering scheme must be maintained regardless of the bit numbering scheme of the target system.

The **DBMDZB** routine should return the first zero bit, starting from left to right. If all bits are one, then a –1 is returned. This function typically uses the FORTRAN **BTEST** function (if one is provided) with appropriate calculations to use the proper bit numbering scheme.

Design Requirements:

None

Error Conditions:

None

3.2. THE SYSTEM GENERATION PROGRAM

After development of the machine dependent source code for the target host machine, the next step in the ASTROS system installation is the assembly of the executable image of the ASTROS system generation program, SYSGEN. The libraries that must be linked to generate this program have been outlined in Section 2 of this manual while this section discusses the function of the SYSGEN program and details the structure of its inputs. These inputs not only define the standard ASTROS system but are also a powerful tool for an advanced user to expand the capabilities of the system. SYSGEN represents one of the most useful features of the ASTROS system architecture in that it provides for automated modification of many of the procedure's capabilities without requiring modification of any existing source code.

The purpose of SYSGEN is to create a system database (SYSDB) defining system parameters through the interpretation of several input files. Also, a FORTRAN routine is written by SYSGEN that provides the link between the ASTROS executive system and the application modules that comprise the run-time library of the procedure. This program unit is then linked with the system during the assembly of the ASTROS executable image. The resultant procedure makes use of the system database as a pool of data that defines the system at run time. These data are

1. the contents of the ASTROS run-time library of MAPOL addressable modules including both utility and application modules, usually delivered as **MODDEF.DAT** or **MODDEF.DATA**;

2. the ASTROS standard executive sequence composed of MAPOL source code statements, usually delivered as **MAPOLSEQ.DAT** or **MAPOLSEQ.DATA**;

3. the set of bulk data entries interpretable by the system and defined through the specification of bulk data templates to be interpreted by the ASTROS Input File Processor (IFP), usually delivered as **TEMPLATE.DAT** or **TEMPLATE.DATA**;

4. the set of relational schemata used by the executive system to satisfy the declaration of relational variables in the MAPOL sequence without forcing the user to explicitly define each schema at run time, usually delivered as **RELATION.DAT** or **RELATION.DATA**; and

5. the set of error message texts from which the **UTMWRT** system message writer utility builds error messages at run time, usually delivered as **SERRMSG.DAT** or **SERRMSG.DATA**.

There is an input file for each of these data which is interpreted by SYSGEN and used to write data to SYSDB in particular formats. These database entities are then used by the ASTROS executive system, application modules and utilities to perform certain functions. Since these program units are designed to interpret the set of data that are present in the SYSDB entities, they are flexible in that virtually any changes to the set of data can be accommodated without modification of the software that uses the data.

The following sections each contain a description of a SYSGEN input file and of the SYSDB database entities that are filled with the corresponding data. These input files contain the definition of the system as developed for the ASTROS procedure. The advanced user may, through the appropriate changes to these inputs, add new modules, add new error messages that may be useful as part of the additional module(s), add new bulk data inputs, add new relational schemata to those that exist or add

new attributes to an existing schema. Finally, the standard solution algorithm itself can be modified, either to include (as a permanent modification) a new feature or to modify an existing capability. The advanced user is cautioned, however, that the standard sequence represents a very tightly interwoven set of functions and any changes should be carefully considered for their ramifications on the multidisciplinary features of the system as it is currently defined.

3.2.1. Functional Module Definition

The functional modules form the computational heart of the ASTROS system. A sequential file contains character data records that define the following information for each module:

1. The name of the module
2. The number of formal parameters
3. If the module is a function or a procedure
4. The type of each formal parameter
5. FORTRAN code lines defining the module call

The purpose of these data is three-fold:

1. To provide the names of modules that are a part of ASTROS
2. To allow the validation of module calls including type checking of the input parameters
3. To define the way in which the results of a module are used and to provide the actual FORTRAN link to activate a module

The format used to provide these data is described in the following section

3.2.1.1 The File Format

The module definition file is organized as a sequence of module entries.

```
MODULE(1) ENTRY
MODULE(2) ENTRY
   ...
```

Each module entry has the following form:

```
MODNAME, NPARM          (A8,I4)
MODTYPE, PARMTYPE(I) (20I4)
   ...
FORTRAN LINES           (A80)
   ...
END                     (A8)
```

where the first line consists of **MODNAME** and **NPARM** and the second line consists of **MODTYPE** and the first 19 **PARMTYPE**s. The **PARMTYPE**s continue, 20 per line, on subsequent lines, as required to supply **NPARM** **PARMTYPE**s.

The lines following the **PARMTYPE**s consist of FORTRAN code (including, but not requiring, comments or blank lines) which implements the module interface to MAPOL.

The last line of a module entry is the word END starting in column 1.

MODNAME is the module name as it appears in MAPOL; 1 to 8 characters beginning with a letter. The name is left-justified in an 8-character field.

NPARM is the number of formal parameters in the calling list of the module. This is a right-justified integer in a 4-digit field. There is a limit of 50 parameters.

MODTYPE is a four-digit code for the module type:
100 means the module is a function with a fixed number of parameters
101 means the module is a function with a variable number of parameters
102 means the module is a procedure

PARMTYPE The declared type of each parameter in the calling list selected from the following codes, each a four-digit integer:

1 Integer
2 Real
3 Complex
4 Logical
5 Not Used
6 Not Used
7 Relational Entity
8 Matrix Entity
9 Unstructured Entity
10 Real, Integer, or Complex

If the **PARMTYPE** is entered as a negative value, the parameter is optional. Note that character **PARMTYPE**s are not supported.

Procedures

For procedures, a call is made to the ASTROS name with a parameter list having symbolic arguments of the correct types. For example, if a module has the following parameters (in order) with the specified data types:

3 Integers
1 Logical
1 Integer
1 Relation
1 Integer
1 Optional integer
1 Optional matrix

1 Matrix

1 Optional matrix

1 Logical

1 Matrix

1 Optional matrix

2 Matrices

1 Unstructured

2 Optional matrices

1 Optional logical

2 Optional matrices

1 Optional unstructured

then the call to this routine is coded as:

```
AROSNSDR  23
 102   1   1   1   4   1   7   1  -1  -8   8  -8   4   8  -8   8   8   9  -8  -8
  -4  -8  -8  -9
C
C     PROCESS 'AROSDR' MODULE - SAERO CONSTRAINT SENS. DRIVER
C
      CALL AROSDR ( IP(1), IP(2), IP(3), LP(4), IP(5), EP(6), IP(7),
     1              IP(8), EP(9), EP(10), EP(11), LP(12), EP(13),
     2              EP(14), EP(15), EP(16), EP(17), EP(18), EP(19),
     3              LP(20), EP(21), EP(22), EP(23) )
END
```

The subscripted array elements are used by the ASTROS executive to pass the actual parameter values. The subscripts must correspond to the order of the arguments in the MAPOL calling list. The following array names are used:

IP – Integer Parameter
RP – Real Parameter
CP – Complex Parameter
LP – Logical Parameter
EP – Entity name

This method passes only scalar parameters to the FORTRAN driver. No mechanism is available to pass FORTRAN arrays.

Functions

For a function, the resultant value is returned to MAPOL on the execution stack. To accomplish this, the programmer ***must*** assign the numeric function result to the FORTRAN variables **IOUT**, **ROUT**, and **COUT** that define the number to the executive. This is analogous to a function in FORTRAN, in which the value must be assigned to the function name within the function unit.

A numeric value is defined as follows:

IOUT(1)	Variable type key with the same definitions as in **PARMTYPE**
IOUT(2,3) or ROUT(2,3) or COUT	Contain the actual integer, real or complex variable value. These arrays are all equivalenced:

If the value is integer, only **IOUT(2)** must be defined. If the value is real, only **ROUT(2)** must be defined. If the value is complex, then either **COUT** must be defined, or **ROUT**(2) and **ROUT(3)** must be defined. What is important is that the data in the second and third words be consistent with the type in **IOUT(1)**.

Further if the function operation depends on the types of arguments (as do the FORTRAN generic functions, e.g. **MAX**, **SIN**), the array

```
TP(I), I=1, NPARAM
```

may be read in the module definition code to determine the type of argument passed. The **PARMTYPE** definition should then be 10 to allow any type to be passed. **TP** uses the same definitions as **PARMTYPE**, except the actual type of the argument is stored. In other words, **TP** contains a 1, 2, or 3 in the location associated with type 10 parameters, depending on the actual type passed in the current call.

For example, if the sine function is desired, the following module definition would be used:

```
SIN         1
 100  10
C
C      SIN - RETURN THE SINE OF THE ARGUMENT
C
   IF( TP(1) .LE. 2 ) THEN
         IOUT(1) = 2
         IF ( TP(1) .EQ. 1 ) RP(1) = IP(1)
         ROUT(2) = SIN(RP(1))
      ELSE
         IOUT(1) = 3
         COUT    = SIN(CP(1))
      ENDIF
END
```

3.2.1.2 SYSGEN Output for Modules

The data defined by the module file are processed and the results are stored on the system database in two entities. The first is a relation called **MODINDEX**. This relation has two attributes: the first, **MODLNAME**, is the module name and the second, **ARGPONTR**, is a pointer to the second entity. This second entity is called **MODLARGS**. Each record of this unstructured entity contains the **MODTYPE** and **PARMTYPE** data from the module definition file for a particular module. Additionally, the output of SYSGEN includes a FORTRAN subroutine called **XQDRIV**. This routine is the module driver for the ASTROS execution monitor. It must be compiled and linked into the system when adding or changing module definitions.

3.2.2. Standard Solution Algorithm Definition

The standard multidisciplinary solution algorithm, in the form of MAPOL source code statements, is contained in a sequential file. The SYSGEN program reads this file and compiles a standard sequence. The results of the compilation are stored on the system database in the form of two relations and an unstructured entity. The first relation is called **&MAPMEM** and has three attributes: **ADDRESS**, **VARTYPE**, and **CONTENT**. This relation stores the execution memory map for the standard MAPOL sequence. **ADDRESS** is an integer containing the address of the variable, **VARTYPE** is an integer denoting the variable type and **CONTENT** is a two-word integer array containing the current value of the variable.

The second relation output from the compilation of the standard sequence is called **&MAPCOD** and has three attributes: **INSSEQ**, **OPCODE**, and **ARGUMENT**. This relation contains the ASTROS machine instructions that represent the compiled MAPOL sequence. **INSSEQ** is an integer containing the instruction sequence number, **OPCODE** is the machine operation code to determine the action to be taken, and **ARGUMENT** is the argument to the operation -- either a memory address of an immediate operand.

The final output from the standard algorithm definition is not directly related to the compilation of the sequence. It is a relation called **&MAPSOU** containing the standard sequence source code statements verbatim. This is stored on the system database, allowing the user to edit the standard sequence to generate a new MAPOL program which directs the ASTROS procedure. The relation has two attributes: **LINENO** and **SOURCE**. **LINENO** is an integer containing the line number and **SOURCE** is a string attribute containing the 80-character source code line.

3.2.3. Bulk Data Template Definition

The ASTROS bulk data decoding module (**IFP**) is driven by templates that are stored on the system database during system generation. The template format for **IFP** was adopted to allow for easy installation of new bulk data entries and for easy modification of existing bulk data entries. The sequential file used by SYSGEN contains the bulk data templates for all the bulk data inputs defined to the ASTROS system in arbitrary order.

3.2.3.1 The File Format

The template definition file has the following format:

```
MAXSET (I8)
NLPTMP (I8)
TEMPLATE 1
TEMPLATE 2
   ...
   ...
```

MAXSET is the maximum number of template sets used to define one bulk data input. Currently, this value is five. **NLPTMP** is the number of lines in each template set. Currently there are six lines in each set. A bulk data template therefore consists of **1,2,3,...MAXSET** template sets, each of which consist of **NLPTMP** template lines which define the structure of the input entry. The definition includes the field size, the field label, the field data type, the field defaults, the field checks, the field database

loading position and, if necessary, a list of relational attributes. The structure of the template set is as follows:

```
BULK DATA ENTRY LABEL
FIELD DATA TYPES
DEFAULT VALUES
ERROR CHECKS
FIELD LOADING POSITION
DATABASE ENTITY DEFINITION
```

A typical bulk data entry template is:

```
CQUAD4 |EID    |PID    |G1     |G2     |G3     |G4     |TM     |ZOFF   |CONT   |
CHAR    INT     INT     INT     INT     INT     INT     INT/REALREAL    CHAR
DEFAULT         EID                                     0.      0.
CHECKS  GT 0    GT 0    GT 0    UG 2    UG 3    UG 4    GT 0
        1       2       3       4       5       6       7       9
CQUAD4  EID     PID1    GRID1   GRID2   GRID3   GRID4   CID1    THETA
+CQUAD4|        |TMAX   |T1     |T2     |T3     |T4     |                       |
CHAR            REAL    REAL    REAL    REAL    REAL
DEFAULT         1.E4
CHECKS          GE 0.   GE 0.   GE 0.   GE 0.   GE 0.
                10      11      12      13      -14
                OFFST0  TMAX    THICK1  THICK2  THICK3  THICK4                $
```

The LABEL *Template Set Line*

The first seven columns of the first template set line define the name of the input data entry. The eighth column is reserved for a field mark, "|". Columns 9 through 72 define the field labels and these are separated by the field mark "|". Columns 73 through 79 indicate to the decoding routines if a continuation card is supported for this entry. On the first template set's **LABEL** line (the template set for the parent line of the bulk data entry), the character string **CONT** indicates that a continuation line is supported. Otherwise, these columns are ignored and a continuation will not be allowed for the entry defined by this template. A continuation template set's **LABEL** line differs from the parent template in that the character string **ETC** can be used in columns 73 through 79 of the continuation template set's **LABEL** line to indicate an open ended entry having a repeating continuation. In this case, the same continuation template will be used to decode all remaining continuation entries. For free format input, the continuation entries must have the input extend to the third field (the second data field). Also, note that ALL template set's **LABEL** lines must have a field mark in column 80 to end the line.

The FIELD DATA TYPE *Template Set Line*

The **FIELD DATA TYPE** Template Set Line defines the types of data that are allowed for each field. Possible data types include: blank, **INT** (integer), **REAL** (real), **CHAR** (character), **INT/REAL** (integer or real), **INT/CHAR** (integer or character), and **REL/CHAR** (real or character). The data type definition characters (i.e., **INT**) must be left justified in the fields.

The DEFAULT *Template Set Line*

The **DEFAULT** template set line defines the default values of the fields. If the input data entry has an empty field, the default value will be used as the input. All values, like the data types, must be left justified in the fields. Three special cases for default values have been incorporated into the decoding routines. The first is the case of a special user input entry to define the defaults for a template. In this case the user supplied values will be substituted for the normal default values. The **BAROR** and **GRDSET** entries are examples of special inputs used to define the defaults. The addition of any other special inputs like these requires program changes in routines **IFPBD**, **BDMERG**, and **IFPDEC**. Another special case is in the referral to another field of the same template to obtain the default value. Referral values can exist for all data types except character (**CHAR**) data. In the case of a default referral, the current template set **LABEL** line is searched for the label referred to and, if the label string is not found, all other template sets, starting at the parent template set, are searched for the string. When the string is found, the corresponding entry field is decoded to obtain the default value. An example of a referral is the **PID** field of the **CQUAD4** card template. The third special case for defaults is the use of a multiplier for an integer default referral. This is only valid for integer type data and the presence of a multiplier is defined by an asterisk "*". For example, **3*NDN**, where **NDN** is the label associated with another integer data field.

The ERROR CHECK *Template Set Line*

The **ERROR CHECK** template set line directs data checking for the decoded fields. Each error check specifies both the type of check and the check value. The available check types depend on the data type (for example, Integer, Real, or Character). Checks that are currently encoded are shown in Table 1.

If additional checks are needed, subroutine **INTCHK** must be modified for integer checks, **RELCHK** must be modified for real checks and **CHRCHK** must be modified for character checks.

When two check values are needed, as for the **IB** and **EB** checks, the first is located on the **ERROR CHECK** template set line and the second is located in the same column position on the **FIELD LOADING POSITION** line.

The FIELD LOADING POSITION *Template Set Line*

The **FIELD LOADING POSITION** template set line is used to place the converted data into the database loading array. The sequence of the numbering is dependent on three conditions. The first is the existence of **CHAR** data on the card. In this case, two hollerith single precision words will be used to store an eight character input and the numbering must account for the two words. The second condition is the sequence of the attribute list for a relational bulk database entity. In this case the loading sequence is determined by the sequence of the attribute list. The third condition occurs when a multiple data type field is present, (e.g., **REL/CHAR**). In this case the first variable type is loaded at the given value and the second variable type is loaded in the next word(s). Again this must be accounted for in the numbering sequence. Finally, when a negative integer value is given as the loading position, the database loading array will be loaded onto the database if input errors have not occurred. A negative value should be used on the field associated with the last attribute of the relational projection. At least one negative loading position must be defined.

Table 1. Bulk DataTemplate Error Checks

CHECK TYPE	DATA TYPE	MEANING
blank	all	No check
GT	Int,Real	Greater than
GE	Int,Real	Greater than or equal
NE	Int,Real	Not equal
LT	Int,Real	Less than
LE	Int,Real	Less than or equal
EQ	All	Equal
EB	Int,Real	Exclusive in between
IB	Int,Real	Inclusive in between
EOR	Int,Real	Either or
GEP	Int,Real	Greater than or equal to the previous value
UG	Int	Unique grid
EUG	Int	Empty or unique grid
COMP	Int	A set of component numbers
MID3	Int	PSHELL MID3 material check
MID4	Int	PSHELL MID4 material check
SA	Int	SPCADD, MPCADD combination check
EIGG	Int	Grid check on the EIGI family of input entries
EIGC	Int	Component check on the EIGI family of input entries
I12	Int	I12 check on the CBAR entry
MATG	Int	E and G check on the MAT1 entry
NU	Real	Check E, G and NU for the MAT1 entry
IDES	Real	Design variable range check
F.T.	Char	Failure theory for the PCOMP entry
YORN	Char	Yes or No
PTYP	Char	Property type for the PLIST entry
ETYP	Char	Element type for the ELIST entry
NORM	Char	Normalization method for the EIGi family of input entries
CMETH	Char	Solution method for the EIGC entry
RMETH	Char	Solution method for the EIGR entry
L.O.	Char	Lamination option for PCOMPi entries
ACMP	Char	Component type for airfoil and CAERO6 entries
BCMP	Char	Component type for BODY and PAERO6 entries
BTYP	Char	Body orientation type for the PAERO6 entry
CONV	Char	Conversion factor quantity type for the CONVERT entry
CTYP	Char	Upper or Lower bound constraint check for DCONXXX, entries
FMETH	Char	Flutter analysis method for the flutter entry
DTYPE	Char	Damping type for the TABDMP1 entry
MPREC	Char	Matrix precision for DMI and DMIG entries
MFORM	Char	Matrix form for DMI and DMIG entries
FFT1	Char	Interpolation method for the FFT entry

CHECK TYPE	DATA TYPE	MEANING
FFT2	Char	Output format selector for the FFT entry
MASSF	Char	Mass matrix form option for the MFORM entry
ETYPL	Char	Element name for the ELEMLIST and DCONTHK entries
CCI	Int	Material property defaulting check for the PCOMP entry
CCR1	Real	Laminae thickness defaulting check for the PCOMP entry
CCR2	Real	Laminae orientation angle defaulting check for the PCOMP entry
CCR3	Real	Laminae orientation angle and thickness defaulting check for PCOMP1 and PCOMP2 entry
GTZOB	Int,Real	Greater than zero or blank
GEZOB	Int,Real	Greater than or equal to zero or blank
ULC	Int	IUST, ILST, and/or ICAM check for the AIRFOIL entry
LAMCHK	Int	DCONLAM/DCONLMN laminate definition check
PLYNORS	Int	DCONLAM/DCONPMN ply definition check
NEBLK	Char	Not equal to blank
ETYPC	Char	Element name for the DCONFTP and DCONTWP entries
PTYPC	Char	Property name for the DCONFTP and DCONTWP entries
ETYPS	Char	Element name for the DCONEP and DCONVM entries
PTYPS	Char	Property name for the DCONEPP and DCONVMP entries
STYPE	Char	Control surface symmetry type
TRIM	Char	TRIM type
UM	Char	UM flag for RBEi entries
TRMACC	Char	Acceleration label check for DCONSCF entry
SCFPRM	Char	Parameter label check for DCONSCF entry
SCFUNIT	Char	Unit label check for DCONSCF entry
VTYPE	Char	Velocity type check for DCONFLT entry
DCNTYP	Char	Constraint type check for DCONLIST entry
FLTFIT	Char	Curve fit type check for FLUTTER entry
LAMCHK	Char	DCONLAM/DCONLMN laminate check

The **DATABASE ENTITY DEFINITION** *Template Set Line*

The **DATABASE ENTITY DEFINITION** template set line names the database entity to be loaded in the first eight columns of the parent template set, and the database attribute list for relational entities (Columns 9-72), of the parent template set. Column 80 is reserved for a map-end character ($). The map-end character indicates the end of the template, and so must occur only on the last database entity definition line for the final set of the template.

3.2.3.2 SYSGEN Output for Template Definitions

The SYSGEN outputs consist of an unstructured entity called **SYSTMPLT** which contains the templates and a relation called **TMPPOINT** which allows efficient access to particular templates. The **SYSTMPLT** entity contains one **RECORD** for each bulk data template in the order it appears in the input template definition file. Therefore, the **RECORD**s are of variable length with the longest **RECORD** containing 80 characters for each of **MAXSET** template sets of **NLPTMP** lines. The **TMPPOINT** relation has two attributes: **CARD** and **RECORD**. **CARD** is an eight character string attribute containing the name of the bulk data entry and **RECORD** is the number where the template is stored in **SYSTMPLT**.

3.2.4. Relational Schema Definition

Each relational database entity requires a SCHEMA that defines its data attributes. A sequential file, containing character data, is used to define these schemata. For each relation there is a list of the attribute names, their types, and, if they are arrays or character data, their length. The details of this file are described below:

3.2.4.1 The File Format

The schema definition file is organized as shown below:

```
REL(1) ENTRY
REL(2) ENTRY
   ...
   ...
REL(NREL) ENTRY
```

Each **RELATION** entry has the following form of free field input. Each input may appear anywhere on the line separated by one or more blanks except "**RELATION**" and "**END**".

```
RELATION RELNAME
ATTRNAME ATTRTYPE ATTRLEN
END
```

where

RELATION	is the keyword "**RELATION**" which signifies that a new **RELATION** schema follows. Must begin in column 1.
RELNAM	is the name of the **RELATION**; it may be one to eight characters beginning with a letter.

ATTRNAME	is the name of the attribute; it may be one to eight characters beginning with a letter.
ATTRTYPE	is the type of the attribute selected from:
	'INT' Integer
	'KINT' Keyed Integer
	'AINT' Array of Integers
	'RSP' Real, single precision
	'ARSP' Array of real, single precision
	'RDP' Real, double precision
	'STR' Character string
	'KSTR' Keyed character string
ATTRLEN	is the optional length of the Attribute. If it is of type **AINT**, **ARSP**, **ARDP**, **STR**, or **KSTR**, the length is not optional. For other types, it should be zero or not present.
END	is the keyword "**END**" which signifies the **END** of the **RELATION** schema.

3.2.4.2 SYSGEN Output for Relations

The data defined by the **RELATION** schema file are processed and the results are stored in two entities on the system database. The first is a **RELATION** called **RELINDEX**. This entity has an attribute **RELTNAME** containing the name of the **RELATION** and an attribute **SCHMPNTR** which is an integer pointer to an unstructured entity called **RELSCHEM**. The **RELSCHEM** entity contains a list of attribute names, types and lengths for each **RELATION**. Each **RELATION** has one tuple in the **RELINDEX RELATION** and one **RECORD** in the **RELSCHEM** entity. Each **RELSCHEM RECORD** consists of a four-word entry for each attribute: two hollerith words containing the attribute name, one hollerith word containing the attribute type and an integer word containing the attribute length.

3.2.5. Error Message Text Definition

The text of ASTROS run time messages is stored and maintained on a sequential file which is used during system generation to create SYSDB entities for use by the ASTROS message writer utility module. There are two reasons for maintaining the message text on an external file (and on SYSDB). First, the storage of message text within the functional modules would use a large amount of memory during execution. Second, storing the messages together in an external file allows for easier maintenance and aids in avoiding needless duplication in message texts. The messages stored on SYSDB from this file are used by the ASTROS utility **UTMWRT** to build error messages during execution.

3.2.5.1 The File Format

The message text file is organized as follows:

*MODULE 1 {**<header>**}

messages

*MODULE 2 {**<header>**}

messages

*MODULE 3 {**<header>**}

...

...

*MODULE <n> {**<header>**}

The **header** is an optional label of any length or content up to 120 characters that typically would describe the relationship among the messages for the specified module. In this way, messages that are logically related (for example, all error conditions from the **IFP** module) can be grouped together for simplified maintenance. The module number <n> is a unique integer identifying the base module number for the group of error messages. It need not be consecutive, which allows for randomly numbered modules.

The format of the message text is as follows:

```
'message text $ more text $ ...'
```

the string is enclosed in a single quotation marks because the message will be used as a character string in a FORTRAN write statement. The $ (dollar sign) is used by the **UTMWRT** utility to place character arguments into the string. For example,

```
'$ ELEMENT $ IS ATTACHED TO SCALAR POINT $.'
```

would appear for CTRMEM element 100 attached to scalar point 1001:

```
CTRMEM ELEMENT 100 IS ATTACHED TO SCALAR POINT 1001.
```

If the user wishes the message to carry over to the next output record, the FORTRAN format **RECORD** terminator (/) can be used *outside* the quotation marks to cause a record advance. For example:

```
'THIS IS ON LINE 1'/' THIS IS ON LINE 2.'
```

Currently there is an implementation limit of 128 characters for the length of the message text after including the arguments. Further details are given in the documentation for the **UTMWRT** utility module.

3.2.5.2 SYSGEN Output for Error Message Text

The data in the message file are used to create two system database entities. The first is an indexed unstructured entity called **ERRMSG**. This entity contains one line of the message text file in each record. The second is an unstructured entity called **ERMSGPNT** which has one record. That record has two words for each module defined in the message file. Those words are the module number **<n>** and the record number of the **ERRMSG RECORD** containing the module header. These are used by the **UTMWRT** to position to the proper message text when called.

3.3. GENERATION OF THE ASTROS SYSTEM

Following the execution of the SYSGEN program the system installation proceeds to the generation of the ASTROS executable image. As indicated in Section 3.2, the SYSGEN program writes a FORTRAN program called **XQDRIV** which must be linked with the remainder of the source code to form the ASTROS system. It is this FORTRAN program which provides the flexible interface between the ASTROS executive and the remainder of the ASTROS modules. To generate the ASTROS system, therefore, it is essential to execute the SYSGEN program as a first step.

The SYSGEN execution is only required once to generate the standard version of ASTROS. This is the version that is defined by the delivered set of SYSGEN input files described in Section 3.2. If, however, the users of the system at a particular installation desire to insert additional modules, the SYSGEN program must be re-executed to recreate the **XQDRIV** submodule. The users may also want to modify other SYSGEN inputs to update the system database to include additional input entries or new relational schema. These changes also require the re-execution of the SYSGEN program (to update the system database) but do not require an update of the ASTROS system. For most purposes, only the module definition file described in Section 3.2.1 requires that the ASTROS executable image be recreated.

4. EXECUTIVE SYSTEM

The ASTROS executive system, as described in Section 3 of the Theoretical Manual, may be viewed as a stylized computer with four components: a control unit, a high level memory, an execution monitor and an Input/Output subsystem. The first three components comprise the executive system modules while the I/O subsystem is embodied in the database. The ASTROS modules that comprise the executive system perform tasks to establish the ASTROS/host interface, initiate the execution and, upon completion of the MAPOL instructions, terminate the execution. These modules also compile the MAPOL sequence, if necessary, and initiate the execution monitor that interprets the MAPOL instructions guiding the execution. This section documents the modules of the ASTROS executive system.

The typical user of ASTROS need not be familiar with the executive system modules since their execution path does not have the flexibility that is available for the engineering modules. The executive modules, however, are important from the viewpoint of the system manager and the program developer for several reasons. First, problems with the machine dependent library on a new host system often show up during the executive modules' initialization tasks. The executive system modules are also important in understanding the treatment of the user's input data stream. To isolate the use of external files to the executive system, for example, the **PREPAS** executive module reads the input data stream and loads those portions that deal with the MAPOL, Solution Control and Bulk Data packets to the database. The system manager, therefore, may find it useful to study the nature of the executive modules and their interrelationships to better understand the implementation of the ASTROS architecture.

Executive System Module: **ASTROS**

Entry Point: **ASTROS**

PURPOSE:

ASTROS is the main entry point for the ASTROS procedure. It controls the execution path through the executive system modules.

MAPOL Calling Sequence:

None

Application Calling Sequence:

None

Method:

The ASTROS routine first sets a flag to tell the executive modules that subsequent calls are associated with the ASTROS procedure rather than with the SYSGEN program. This flag is required since the compiler and other executive routines are shared between the two programs and require sightly different execution paths. The machine dependent initialization routine, **XXINIT**, is then called to perform the initialization tasks required on the current host system. The initialization is completed by zeroing the execution monitor's stack length, calling the machine independent initialization routine, **XQINIT**, labeling the output listing and the starting the timing summary.

The **PREPAS** module is then called to read the user's input data stream. On return from **PREPAS**, the MAPOL compiler is called if a MAPOL compilation is required. Finally, the execution monitor, **XQTMON**, is called to interpret the ASTROS machine instructions representing the compiled MAPOL sequence. All subsequent activities in the ASTROS execution are controlled by this module until all the MAPOL instructions have been completed. Upon return from **XQTMON**, the main driver terminates the execution by writing the closing label, calling the **XQENDS** module to close the database, dumping the timing summary and performing any other closing tasks.

The engineering modules (addressed by the execution monitor) may also terminate the execution of the system. In these cases, the general application utility module, **UTEXIT**, is used since this routine assumes that an error exit has occurred. **UTEXIT**, however, also calls the **XQENDS** executive module to assure clean termination of all executions.

Design Requirements:

None

Error Conditions:

None

Executive System Module: **XQINIT**

Entry Point: **XQINIT**

PURPOSE:

To perform machine independent system initialization tasks.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL XQINIT
```

Method:

This routine completes the page titling information on the **TITLE** line used by the **UTPAGE** utility. The ASTROS version number is placed in **TITLE** (which is in the **/OUTPUT2/** common block) in characters 88 through 107. The current date is obtained using **XXDATE** and placed in characters 109 to 117. The page number label is then placed in characters 120 to 121. Thus, the page number itself is left to fit in characters 123 to 128.

Design Requirements:

None

Error Conditions:

None

Executive System Module: **PREPAS**

Entry Point: **PREPAS**

PURPOSE:

To perform the first pass through the user's input data stream, to initialize the open core memory manager and to attach the scratch and system databases.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL PREPAS ( MAPFLG, SOLFLG, BLKFLG )
```

MAPFLG	Integer flag denoting the presence or absence of a MAPOL packet in the input data stream (Output) = **0** if no MAPOL packet = **1** if a MAPOL packet exists in the input stream
SOLFLG	Integer flag defined like **MAPFLG** denoting the presence or absence of a Solution Control packet in the input data stream (Output)
BLKFLG	Integer flag defined like **MAPFLG** denoting the presence or absence of a Bulk Data packet in the input data stream (Output)

Method:

This routine performs the first pass over the user's input data stream, performs actions based on the **ASSIGN DATABASE**, **DEBUG** and MAPOL **EDIT** command inputs and prepares MAPOL, solution control and bulk data inputs for access by the appropriate modules. The order of operations is crucial and can be summarized as follows:

1. The debug packet must be processed first since the debug flags can affect the memory management system.
2. Immediately following the debug packet, the memory manager must be initialized, and the scratch (run-time) database (based on the **ASSIGN DATABASE** entry) and the system database must be attached, ***in that order***. The order is dictated by the requirement that the memory manager be available for the **DBINIT** calls and the scratch database must be allocated before the system database. The system database must be allocated in order to process the MAPOL packet (which may apply **EDIT** operations to the standard MAPOL sequence stored on the system database).

Once the **ASSIGN DATABASE** and debug packets are processed, the remaining packets could be ordered in any fashion, but a fixed sequence of MAPOL, solution control and bulk data packets has been imposed.

The procedure used in **PREPAS** is to read the input stream one 80-character record at a time. The first nonblank records must be the **ASSIGN DATABASE** entry. The corresponding records are set aside in **ASNCRD** for use after the debug packet is processed. After the **ASSIGN DATABASE** entry has been encountered, each input record is read and searched to see if one of the input stream keywords appears in the first eight characters following the first nonblank character. The keywords are:

1. **DEBUG** denoting the start of the debug packet
2. **MAPOL** denoting the start of the MAPOL packet containing a complete new MAPOL sequence

3. **EDIT** denoting the start of the MAPOL packet containing edit commands to be applied to the standard sequence
4. **SOLUTION** denoting the start of the solution control packet
5. "**BEGIN**_" denoting the start of the bulk data packet. Note the trailing blank after **BEGIN** and the absence of the optional **BULK** keyword.
6. **ENDDATA** denoting the end of the bulk data packet
7. **INCLUDE** naming the secondary file from which to read the input

Note that all the keywords except **ENDDATA** and **INCLUDE** mark the beginning of a new packet. The **INCLUDE** keyword does not change the current packet and **ENDDATA** marks the end of the valid input. If the current record is one of the keyword records, flags are set to indicate that a new packet has been initiated or, for **INCLUDE**, the include file is opened and processing continues with the new input file until it is exhausted. Records that are not keyword records are processed as follows:

1. **DEBUG** packet records are sent to the **CRKBUG** utility to interpret the debug commands and set the executive system debug command flags in the **/EXEC02/** common and set the other debug command flags by **UTSFLG** to activate run time debugs.
2. MAPOL packet records representing a replacement MAPOL sequence are written to the unstructured entity **&MAPLPKT** for processing in the MAPOL module.
3. MAPOL packet records representing an **EDIT** of the standard sequence are passed to the **MAPEDT** subroutine to be interpreted. The resultant MAPOL sequence is written to the unstructured entity **&MAPLPKT** for processing in the MAPOL module.
4. **SOLUTION** packet records are written to the unstructured entity **&SOLNPKT** for processing in the **SOLUTION** module.
5. Bulk Data packet records are written to the unstructured entity **&BKDTPKT** for processing in the IFP module.

Design Requirements:

None

Error Conditions:

1. Input stream does not begin with an **ASSIGN DATABASE** entry.
2. An input stream keyword appears out of order.
3. An **ENDDATA** statement appears outside the bulk data packet.
4. No filename was found on an **INCLUDE** statement.
5. **INCLUDE** file cannot be opened.
6. Input record lies outside any input packet (typically following an **ENDDATA**).
7. FORTRAN read error on the primary input stream or included file.
8. An **INCLUDE** statement appearing in a included file.

Executive System Module: **MMINIT**

Entry Point: **MMINIT**

PURPOSE:

To initialize the memory manager.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MMINIT ( SIZE )
```

SIZE	The number of single precision words in open core (Integer, Input)

Method:

This routine establishes the initial block headers for open core memory. A block header is written representing one free block of **SIZE** words less those required for the block header. The block header is either six or eight words depending on whether the **MEMORY** debug has been selected by the user in the input stream. The size must correspond to the actual declared length of the open-core common block **/MEMORY/**.

Design Requirements:

None

Error Conditions:

None

Executive System Module: **DBINIT**

Entry Point: **DBINIT**

PURPOSE:

To initialize the processing for a database.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBINIT ( DBNAME, PASSWD, STAT, RW, USRPRM )
```

DBNAME	The database name (Character, Input)
PASSWD	The database password (Character, Input)
STAT	The database status (Character, Input) One of **OLD**, **NEW**, **SAVE**, **PERM** or **TEMP**
RW	Read/Write status (Character, Input) One of **RO** or **R/W**
USRPRM	Installation dependent user parameters (Character, Input)

Method:

This routine opens the named database for access. It performs any machine and installation dependent processing by accessing the database machine dependent library routines **DBMDCX** and **DBMDIX**. All the in-core buffers required for subsequent database access are allocated using the database memory management routines.

Design Requirements:

1. The first call to **DBINIT** must define the run-time or scratch database. Any other databases may then be initialized.

Error Conditions:

1. Any error conditions on the file operations occuring in **DBINIT** will terminate the execution.

Entry Point: **DBCINI**

PURPOSE:

To initialize the processing for a database and return a status code rather than terminate on error.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBCINI ( DBNAME, PASSWD, STAT, RW, USRPRM, ISTAT )
```

DBNAME	The database name (Character, Input)

PASSWD	The database password (Character, Input)
STAT	The database status (Character, Input) One of **OLD**, **NEW**, **SAVE**, **PERM** or **TEMP**
RW	Read/Write status (Character, Input) One of **RO** or **R/W**
USRPRM	Installation dependent user parameters (Character, Input)
ISTAT	Return status

1	Duplicate database name
2	Too many databases open
3	Bad RW parameter
4	Index file block size too small
5	Bad data in file found on old open
6	Password check failure on old open
7	Old formatted database not supported
8	Read only open on new database illegal
9	Bad STAT parameter
100	Values are machine dependent - see **DBMDIX**

Method:

This routine opens the named database for access. It performs any machine and installation dependent processing by accessing the database machine dependent library routines **DBMDCX** and **DBMDIX**. All the in-core buffers required for subsequent database access are allocated using the database memory management routines.

Design Requirements:

1. The first call to **DBINIT** must define the run-time or scratch database. Any other databases may then be initialized.

Error Conditions:

1. Any error conditions on the file operations occuring in **DBCINI** will be flagged using the **ISTAT** parameter and control returned to the calling routine.

Executive System Module: **MAPOL**

Entry Point: **MAPOL**

PURPOSE:

To compile a MAPOL program.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MAPOL
```

Method:

The input MAPOL program or a MAPOL program representing a modified standard sequence is read from the **&MAPLPKT** unstructured entity and compiled. The resultant machine code instructions and memory map are written to the **MCODE** and **MEMORY** entities for use by **XQTMON** in executing the MAPOL program.

Design Requirements:

None

Error Conditions:

1. MAPOL syntax errors
2. Illegal argument types used in MAPOL module calls

Executive System Module: **XQTMON**

Entry Point: **XQTMON**

PURPOSE:

To execute a set of ASTROS machine instructions representing a MAPOL program.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL XQTMON
```

Method:

If no MAPOL packet was in the input stream, **XQTMON** copies the standard sequence machine instructions and memory map from the system database into the **MCODE** and **MEMORY** entities. If a MAPOL compilation took place, the current sequence's data are already in **MCODE** and **MEMORY**. Beginning with the first instruction, which is passed to **XQTMON** from the MAPOL compiler or retrieved from the system database, the machine instructions are executed by this module.

Most instructions are processed directly by the **XQTMON** module; for example, stack operations, entity creations and scalar arithmetic operations. If the instruction is a module call, however, the **XQDRIV** executive subroutine, previously written by the SYSGEN program, is called to access the MAPOL module to which the machine instruction refers.

Design Requirements:

None

Error Conditions:

MAPOL run time errors

Executive System Module: **XQENDS**

Entry Point: **XQENDS**

PURPOSE:

To cleanly terminate the ASTROS execution.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL XQENDS
```

Method:

The executive module's database entities that remain open throughout the ASTROS execution are closed and the **DBTERM** executive module is called to close the database(s).

Design Requirements:

None

Error Conditions:

None

Executive System Module: **DBTERM**

Entry Point: **DBTERM**

PURPOSE:

To terminate processing of all open database.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBTERM ( DBNAME )
```

DBNAME	Database name or blank (Character, Input)

Method:

The database entity name table (**ENT**) is searched and all open entities corresponding to **DBNAME** are closed and the **ENT** deleted. If **DBNAME** is blank, all open entities on all databases are closed but the **ENT**s are not deleted. If all databases are to be closed, the database name table (**DBNT**) is searched and all in-core buffers are freed. The first record of each database is updated to indicate that it was properly closed and any system dependent termination is performed. If a particular database is to be closed, these operations are done only for the named database. Finally, if all databases are closed, the **ENT**, **DBNT** and the name substitution table (**NST**) are released at the close of the **DBTERM**.

Design Requirements:

None

Error Conditions:

None

5. ENGINEERING APPLICATION MODULES

The modules documented in this section fall under the category of engineering application modules. These modules constitute the majority of the ASTROS software and do the tasks necessary to perform the analyses supported by the ASTROS system. Unsurprisingly, the difference between an "engineering application" module and other modules in the ASTROS system is not always clear. The most useful definition may be that an engineering application module is one that does not fall into any other category. They do, however, share some common attributes that can be used to help distinguish them from other modules. First, an engineering application module has no application calling sequence: it is only accessible through the executive system. A related attribute is that no engineering module may be called by another module, whereas utility modules may be called by other modules or by the executive system. Finally, an engineering application module is one that establishes an open core base address by calling the **MMBASE** and/or **MMBASC** utilities and uses that one base address throughout its execution.

The following subsections document each of the engineering application modules that comprise the ASTROS system. Each module is documented using the standard format shown in the introduction, but some additional comments are necessary. First, the MAPOL language allows the use of optional arguments in the calling sequences. This feature has been used in many modules to provide optional print selections or to allow the module to be used in slightly different ways. This is particularly true for the matrix reduction "modules" (**GREDUCE**, **FREDUCE** and **RECOVA**) which may almost be considered utilities. When the argument in the MAPOL calling sequence is optional, it is so indicated in the calling list. The **Method** section then describes the alternative operations that take place depending on the presence of the optional argument.

A second point to emphasize is the general nature of the **Method** sections for the engineering module documentation. In no way does this documentation attempt to lead the reader through the code segments of the module. Instead, a general descripton of the algorithm is given which, in combination with the in-line comments, should give the programmer an adequate understanding of the module. The system programmer wanting to make extensive software modifications to existing modules will still need to study the actual code segments in some detail. The level of detail in the engineering module documentation is considered more appropriate for the ASTROS analyst/designer who wants to understand how ASTROS uses the existing pool of engineering modules and to know the "initial state" that the module expects to exist when it is called. The analyst may then make "nonstandard" use of the module to perform alternative analyses. These, therefore, are the data emphasized in the module documentation that follows.

A final note should be made relative to the description of the MAPOL module calling sequences. Version 12 of ASTROS has introduced user defined boundary condition identification numbers, called **BCID** in this chapter, which are used when specifying user defined functions. These are not to be confused with the boundary condition index, or subscript, which is a sequential counter used in MAPOL. This counter is shown as the entity subscript **BC**.

Engineering Application Module: **ABOUND**

Entry Point: **ABOUND**

Purpose:

To generate flags for the current boundary condition that indicate which constraint types are active. These are then returned to the executive sequence to direct the execution of the required sensitivity analyses.

MAPOL Calling Sequence:

```
CALL ABOUND ( NITER, BCID, CONST, NDV, ACTBOUND, NAUS, NACSD, [PGAS], PCAS,
              PRAS, ACTAERO, ACTDYN, ACTFLUT, ACTPNL, ACTBAR, NMPC,
              NSPC, NOMIT, NRSET, NGDR, USET(BC) );
```

NITER	Design iteration number (Integer, Input)
BCID	User defined boundary condition identification number (Integer, Input)
CONST	Relation of constraint values (Character, Input)
NDV	Number of design variables (Integer,Input)
ACTBOUND	Logical flag, **TRUE** if boundary condition is active (Logical,Output)
NAUS	Number of active STATICS displacement vectors (Integer, Output)
NACSD	Number of active STATICS stress and/or displacement constraints. (Integer, Output)
[PGAS]	Partition vector for active STATICS displacement vectors (Output)
PCAS	Unstructured entity which contains the unique STATICS subcase numbers for the displacement dependent STATICS constraints that are active for the boundary condition. Only constraints for the current boundary condition are included in the list (Output)
PRAS	Unstructured entity which contains the unique STATICS subcase numbers for the response functions that are required by active user function constraints (Output)
ACTAERO	Logical flag, **TRUE** if any active aeroelastic effectiveness constraints and/or responses.(Logical,Output)
ACTDYN	Logical flag, **TRUE** if any active frequency constraints and/or responses (Logical,Output)
ACTFLUT	Logical flag, **TRUE** if any active flutter constraints and/or responses (Logical,Output)
ACTPNL	Logical flag, **TRUE** if any active panel buckling constraints (Logical,Output)
ACTBAR	Logical flag, **TRUE** if any active Euler buckling constraints (Logical,Output)
NMPC	Number of **MPC** degrees of freedom (Integer, Output)
NSPC	Number of **SPC** degrees of freedom (Integer, Output)
NOMIT	Number of omitted DOF (Integer, Output)

NRSET	Number of support DOF (Integer, Output)
NGDR	Denotes dynamic reduction in the boundary condition (Integer, Output) = **0** No **GDR** = **-1** **GDR** is used
USET(BC)	The unstructured entity defining structural sets for each degree of freedom (Character, Input), where **BC** represents the MAPOL boundary condition loop index number.

Application Calling Sequence:

None

Method:

The module first reads the **CONST** relation for active constraints associated with the boundary condition. If any entries are found, the **ACTBOUND** flag is set to on. If not, control is returned to the executive.

The **CASE** relation is then read for all subcases associated with the boundary condition. The number of **STATICS** subcases is counted in preparation for determining the partitioning vector of active subcases and the counts of right-hand-sides and constraints that will determine if the gradient or virtual load method will be used in sensitivity analysis. **SAERO** disciplines cannot use the virtual load method and are therefore not treated in this manner.

Then the **USET** and **CASE** entities are searched to set the boundary condition flags that are output to control the reduction processes during the sensitivity phase. Then the work of the module begins.

The active subcases and constraint types are determined for each of the entries in **CONST** that were read. During the pass through the active constraint set, the partitioning vector for the STATICS displacement matrix is built (of the number of right-hand-sides columns, only the **NAUS** columns will be active). For **STATICS** constraints that are dependent on the displacement vector derivatives, the active subcase is identified and the partitioning vector, **PGAS**, and the set of subcase ids that are active, **PCAS**, are loaded. Any subcases which have displacement or element stress/strain response functions required because there are active user functional constraints, are also defined as active. The partitioning vector, **PGAS**, and the set of subcase numbers that are active, **PRAS**, are loaded.

Then a summary of all the active constraints for the boundary condition is echoed to the output file.

Design Requirements:

1. This module must follow the complete analysis phase for all the boundary conditions and is the first module called within the sensitivity boundary condition loop.

Error Conditions:

None

Engineering Application Module: **ACTCON**

Entry Point: **ACTCON**

Purpose:

To determine whether the design task has converged. If the optimization has not converged, this module selects which constraints are to be included in the current redesign. On print request, this routine computes and prints the values of the local design variables.

MAPOL Calling Sequence:

```
CALL ACTCON ( NITER, MAXITER, NRFAC, NDV, GLBDES, LOCLVAR, [PTRANS], EPS,
              APPCNVRG, GLBCNVRG, CTL, CTLMIN, CONST, [AMAT], DESHIST,
              PFLAG, OLOCALDV );
```

NITER	The current iteration number (Integer, Input)
MAXITER	The maximum number of allowable iterations (Integer, Input)
NRFAC	Determines the minimum number of retained constraints equal to **NRFAC*NDV** (Real, Input)
NDV	The number of global design variables (Integer, Input)
GLBDES	Relation of global design variables (Character, Input)
LOCLVAR	Relation containing the relationship between local variables and global variables in the design problem (Character, Input)
[PTRANS]	The design variable linking matrix (Character, Input)
EPS	A second criteria for constraint retention. All constraints greater than or equal to **EPS** will be retained (Real, Input)
APPCNVRG	The approximate problem converge flag from module **DESIGN** or **FSD** (Logical, Input) = **FALSE** if not converged = **TRUE** if no change in objective function value and design variables
GLBCNVRG	Final convergence flag (Logical, Output) = **FALSE** if not converged = **TRUE** if global converge was achieved
CTL	Constraint tolerance for active constraints (Real, Input)
CTLMIN	Constraint tolerance for violated constraints (Real, Input)
CONST	Relation of constraint values (Character, Input)
[AMAT]	The matrix entity of constraint gradients (Output)
DESHIST	Relation of design iteration information (Character, Input)
PFLAG	The logical flag to indicate if design model punching is requested for the current iteration (Logical, Input)
OLOCALDV	Relation to store the output local design variables requested in solution control (Character, Input)

Application Calling Sequence:

None

Method:

The initial action of the **ACTCON** module is to flush the **[AMAT]** matrix of constraint gradients for the sensitivity analysis that is to follow. This erases the constraint sensitivities from the previous design iteration and prepares the matrix to be loaded by the constraint sensitivity evaluation modules. Following this bookkeeping task, the **ACTCON** module begins the process of selecting the active constraints for the next redesign cycle. The first computation of the number of retained constraints is done using the value **NRFAC*NDV**. This represents a minimum number of constraints to retain. The vector of current constraint values is brought into core and sorted. Then the **EPS** value and initial number of retained constraints are used to determine the cutoff value for the active constraints. This cutoff value, **CMIN**, will either be the constraint value such that **NRFAC*NDV** constraints are retained, the constraint value closest to, but less than, **EPS** or the minimum constraint value if there are fewer than **NRFAC*NDV** constraints. During this phase the count of thickness constraints that are retained even though they do not satisfy the **NRFAC** and **EPS** retension criteria is kept. A summary is printed that indicates the number of constraints kept for each reason: **NRFAC**, **EPS** and **DCONTHK**.

If the approximate problem convergence flag, **APPCNVRG**, is **TRUE**, the maximum constraint value is tested to determine if global convergence has been achieved based on **CTL** and **CTLMIN**. The **GLBCNVRG** flag is set to **TRUE** if global convergence has been reached.

The next task of the **ACTCON** module is to set the active constraint attribute in the **CONST** relation. This is done by retrieving each tuple of the **CONST** relation and comparing the constraint value against the cutoff value, **CMIN**. The appropriate constraints are then marked active by setting the **ACTVFLAG** attribute to unity. Finally, the **ACTCON** module prints out the results of the design process if global convergence or the maximum number of iterations has been reached. This includes the print of the design iteration history and, if requested by Solution Control, the summary of global and local design variables.

Design Requirements:

1. **ACTCON** must be the first module called following the analysis phase of the optimization segment of the standard sequence. That is, it follows all the analysis boundary conditions but precedes the sensitivity evaluations.

Error Conditions:

1. No design constraints have been applied in the optimization problem.

Engineering Application Module: **AEROEFFS**

Entry Point: **AROSEF**

Purpose:

Evaluates aeroelastic effectiveness sensitivities.

MAPOL Calling Sequence:

```
CALL AEROEFFS ( NITER, BCID, SUB, SYM, NDV, CONST, PCAE, [EFFSENS], [AMAT] );
```

NITER	Current iteration number (Input, Integer)
BCID	User defined boundary condition identification number (Integer, Input)
SUB	Current static aeroelastic subscript number (Input, Integer)
SYM	Symmetry flag for the current call. Either 1 for symmetric or -1 for anti-symmetric (Input, Integer)
NDV	Number of global design variables (Input,Integer)
CONST	Relation of design constraints (Input)
PCAE	Unstructured entity containing information indicating which pseudodisplacements (displacements due to unit configuration parameters) are active for the current design iteration (Input)
EFFSENS	The matrix of dimensional stability derivative sensitivities (Input)
AMAT	The matrix of constraint gradients (Output)

Method:

The **CASE** relation is read first to retrieve the **SUPPORT** set for the current boundary condition. The number and location of the support DOF are returned from the utility routine **SEFCHK**. Then the **CONST** relation is read for active lift effectiveness (**DCONCLA**), aileron effectiveness (**DCONALE**) and stability coefficient constraints (**DCONSCF**) for the current boundary condition, subscript and iteration.

The **EFFSENS** matrix, of dimension **NSUP*NDV*NAUE** where **NSUP** is the number of support dofs and **NAUE** is the number of active pseudodisplacement fields of the set computed in **SAERO** for the applied constraints.

The whole **EFFSENS** matrix is read into memory and then the loop over active constraints begins. For each active constraint, the **DISPCOL** attribute of the **CONST** relation is used to determine which column of pseudodisplacements is associated with the constraint. The **PCAE** entity is then used to determine which column of the reduced set of active pseudodisplacement fields is the proper column. Once located, the constraint sensitivities may be computed from the dimensional stability coefficient derivatives and the normalization data stored in the **CONST** relation in the **SAERO** module. The constraint derivatives are computed from the following relationships.

The flexible stability coefficient response sensitivities which are required by the active user function constraints are also computed in this module. Those sensitivities are stored into relational and matrix entities to be used by the user function evaluation utilities.

Lift Effectiveness:

Upper Bound

```
CLAREQ > 0.0
   DG/DX =  SENS ROW / (CLA       * CLAREQ )
                              RIGID

CLAREQ < 0.0
   DG/DX =  -SENS ROW / (CLA      * CLAREQ )
                               RIGID

CLAREQ = 0.0
   DG/DX =  SENS ROW / CLA
                               RIGID
```

Lower Bound

```
CLAREQ > 0.0
   DG/DX =  -SENS ROW / (CLA      * CLAREQ )
                               RIGID

CLAREQ < 0.0
   DG/DX =  SENS ROW / (CLA       * CLAREQ )
                              RIGID

CLAREQ = 0.0
   DG/DX =  -SENS ROW / CLA
                              RIGID
```

where CLA_{RIGID} is stored in the **SENSPRM1** attribute of **CONST** and **CLAREQ** is stored in the **SENSPRM2** attribute of **CONST**

Aileron Effectiveness:

Upper Bound

```
AEREQ > 0.0
   DG/DX =(-SENS * CMXP + SENS * CMXA ) / (AEREQ *  CMXP **2
              1      FLX    2      FLX                FLX

AEREQ < 0.0
   DG/DX =( SENS * CMXP - SENS * CMXA ) / (AEREQ *  CMXP **2
              1      FLX    2      FLX                FLX

AEREQ = 0.0
   DG/DX =(-SENS * CMXP + SENS * CMXA ) / CMXP ** 2
              1      FLX    2      FLX     FLX
```

Lower Bound

```
AEREQ > 0.0
   DG/DX =( SENS * CMXP - SENS * CMXA ) / (AEREQ *  CMXP **2
              1      FLX    2      FLX                FLX

AEREQ < 0.0
   DG/DX =(-SENS * CMXP + SENS * CMXA ) / (AEREQ *  CMXP **2
              1      FLX    2      FLX                FLX

AEREQ = 0.0
   DG/DX =( SENS * CMXP - SENS * CMXA ) / CMXP ** 2
              1      FLX    2      FLX     FLX
```

where $CXMA_{FLEX}$ is stored in the **SENSPRM1** attribute of **CONST** and $CMXP_{FLEX}$ is stored in the **SENSPRM2** attribute of **CONST** and **2.0*AEREQ/(57.3*REFB)** is in **SENSPRM3**

Stability Coefficient:

Upper Bound

```
REQ > 0.0
   DG/DX =  SENS ROW / REQ
REQ < 0.0
   DG/DX =  -SENS ROW / REQ
REQ = 0.0
   DG/DX =  SENS ROW
```

Lower Bound

```
REQ > 0.0
   DG/DX =  -SENS ROW / REQ
REQ < 0.0
   DG/DX =  SENS ROW / REQ
REQ = 0.0
   DG/DX =  -SENS ROW
```

where **REQ**, the dimensional required value is stored in the **SENSPRM1** attribute of **CONST**

The rows of **EFFSENS** associated with each constraint are dependent on the constraint type in the following way:

(1) Lift Effectiveness constraints always use the plunge DOF

(2) Aileron Effectiveness constraints always use the roll DOF

(3) Stability Coefficient constraints always use the row associated with the constrained axis. The constrained axis number (1,2,3,4,5,6) is stored in real form in the **SENSPRM2** attribute of **CONST**.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **AEROSENS**

Entry Point: **AROSNS**

Purpose:

To compute the sensitivities of the rigid body accelerations and aerodynamic performance parameters (**DCONTRM**) for active steady aeroelastic subcases associated with the current subscript.

MAPOL Calling Sequence:

```
CALL AEROSENS ( NITER, BCID, MINDEX, SUB, CONST, SYM, NDV, BGPDT(BC), STABCF,
                [PGAA], [LHSA(BC,SUB)], [RHSA(BC,SUB)], [DRHS], [AAR],
                [DDELDV], [AMAT] );
```

NITER	Design iteration number (Integer, Input)
BCID	User defined boundary condition identification number (Integer, Input)
MINDEX	Mach number index for the boundary condition to recover the proper stability coefficient data (Integer, Input)
SUB	The subscript identifier for the current **SAERO** subcases (Integer, Input)
CONST	Relation of constraint values (Character, Input)
NDV	The number of global design variables (Integer, Input)
SYM	The symmetry flag for the current **SAERO** subcases (Integer, Input)
BGPDT(BC)	Relation of basic grid point coordinate data (Character, Input), where **BC** represents the MAPOL boundary condition loop index number
STABCF	Relation of rigid stability coefficient data (Character, Input)
[PGAA]	Partitioning vector used to obtain g-set active displacement and acceleration vectors for all static aero subcases that have active trim parameter, stress, strain and/or displacement constraints. (Input)
[LHSA(BC,SUB)]	Modified inertia matrix (Character, Input), where **BC** represents the MAPOL boundary condition loop index number
[RHSA(BC,SUB)]	Modified applied load matrix (Character, Input), where **BC** represents the MAPOL boundary condition loop index number
[DRHS]	Matrix entity containing the sensitivity of **[RHSA]** to the design variables (Character, Input)
[AAR]	Matrix entity containing the sensitivities of structural accelerations either zero (for fixed accelerations) or from solution of **LHSA*AAR = RHSA*DDELDV + DRHS** (Output)
[DDELDV]	Matrix entity containing the sensitivity of the configuration parameters to the design variables. Either zero (for **FIXED** control parameters) or from the solution of **LHSA*AAR = RHSA*DDELDV + DRHS** (Output)
[AMAT]	Matrix entity containing the sensitivities of the active aeroelastic control parameter (**DCONTRM**) constraints to the design variables (Character, Output)

Application Calling Sequence:

None

Method:

First the **CASE** relation is read for the **SAERO** subcases in the boundary condition. Then the **STABCF** entity is read for the terms associated with the current **MINDEX**. Then the **TRIM**, control linking and control effectiveness data are read. Finally, the **CONST** relation for the active **DCONTRM**, stress and displacement constraints associated with the current subscript value are read into memory. Then the number of trim subcases (active/associated with **SUB**) is determined and the **PGAA** matrix is read and the number of active subcases is determined. The number of columns in the **DRHS** matrix (=**NDV***number of active subcases for this **SUB** value) is determined.

At this point, an trim solution very similar to the one done in the **SAERO** analysis module is performed to solve for the **AAR** rigid body acceleration derivatives and the **DDELDV** trim parameter sensitivities. The **DRHS** matrix is difficult to deal with since it must be partitioned for each subcase to just the **NDV** columns associated with the subcase under consideration. (Just as in **SAERO**, each subcase must be solved for independently since the effectiveness and control linking are subcase dependent.) Given the correct **NDV** columns in **DRHS**, the following matrix expression is available:

$$\begin{bmatrix} \mathbf{LHS}_{ff} & \mathbf{LHS}_{fk} \\ \mathbf{LHS}_{kf} & \mathbf{LHS}_{kk} \end{bmatrix} \begin{bmatrix} \mathbf{AR}_{free} \\ \mathbf{AR}_{known} \end{bmatrix} = \begin{bmatrix} \mathbf{RHS}_{fu} & \mathbf{RHS}_{fs} \\ \mathbf{RHS}_{ku} & \mathbf{RHS}_{ks} \end{bmatrix} \begin{bmatrix} \mathbf{DEL}_{u} \\ \mathbf{DEL}_{s} \end{bmatrix} + \begin{bmatrix} \mathbf{DRHS}_{f} \\ \mathbf{DRHS}_{k} \end{bmatrix}$$

Where:	Represents:
F+K	Number of **SUPORT** point DOF
F	Set of free accelerations, **AR**
K	Set of known(**FIXED**) accelerations, **AR**
U+S	Number of **AERO** parameters
U	Set of unknown parameters
S	Set of set(**FIXED**) parameters
Note that $\mathbf{AR}_{known}$ and $\mathbf{DEL}_{s}$ sensitivities are zero by definition.	

These equations must be rearranged to get free accelerations and unknown delta's on the same side of the equation:

$$\begin{bmatrix} \mathbf{LHS}_{ff} & -\mathbf{RHS}_{fu} \\ \mathbf{LHS}_{kf} & -\mathbf{RHS}_{ku} \end{bmatrix} \begin{bmatrix} \mathbf{AR}_{free} \\ \mathbf{DEL}_{u} \end{bmatrix} = \begin{bmatrix} -\mathbf{LHS}_{fk} & \mathbf{RHS}_{fs} \\ -\mathbf{LHS}_{kk} & \mathbf{RHS}_{ks} \end{bmatrix} \begin{bmatrix} \mathbf{AR}_{known} \\ \mathbf{DEL}_{s} \end{bmatrix} + \begin{bmatrix} \mathbf{DRHS}_{f} \\ \mathbf{DRHS}_{k} \end{bmatrix}$$

We must handle the degenerate case where all accelerations or all delta's are known. Once the solution is obtained, the free acceleration derivatives and unknown trim parameter derivatives are unscrambled and loaded into subcase specific **AAR** and **DDELDV** entities.

Finally, if any active **DCONTRM** constraints exist, the **AAR** or **DDELDV** matrix for the current subcase is used to compute the **AMAT** terms for them.

Upper bound

```
REQ > 0.0
   DG/DX =  SENS / REQ
REQ < 0.0
   DG/DX =  - SENS / REQ
REQ = 0.0
   DG/DX =  SENS
```

Lower bound

```
REQ > 0.0
   DG/DX =  - SENS / REQ
REQ < 0.0
   DG/DX =  SENS / REQ
REQ = 0.0
   DG/DX =  - SENS
```

Where **REQ** is stored in the **SENSPRM1** attribute of **CONST** and **SENS** is the raw acceleration or deflection sensitivity.

The final operation for the subcase is to merge the **NDV AAR** and **DDELDV** columns for the current subcase into the output matrices. The output matrices have **NDV** columns for each active subcase in subcase order of **SAERO** disciplines in the **CASE** relation.

The trim parameter response sensitivities which are required by the active user function constraints are also computed in this module.

Design Requirements:

1. This module assumes that either strength and/or **DCONTRM** constraints exist for the static aeroelastic analyses in the current boundary condition.

Error Conditions:

None

Engineering Application Module: **AMP**

Entry Point: **AMP**

Purpose:

To compute the discipline dependent unsteady aerodynamic matrices for flutter, gust and blast analyses.

MAPOL Calling Sequence:

```
CALL AMP ( [AJJTL], [D1JK], [D2JK], [SKJ], [QKKL], [QKJL], [QJJL], [AJJDC] );
```

[AJJTL]	Matrix containing the list of AIC matrices for each Mach number, reduced frequency and symmetry option in transposed form (Input)
[D1JK]	Real part of the substantial derivative matrix (Input)
[D2JK]	Imaginary part of the substantial derivative matrix (Input)
[SKJ]	Integration matrix (Input)
[QKKL]	Matrix list containing the matrix product: **[SKJ]*[TRANS(AJJT)]$^{-1}$*([D1JK] + ik[D2JK])** used for flutter and gust analyses (Output)
[QKJL]	Matrix list containing the matrix product: **[SKJ] * [TRANS(AJJT)]$^{-1}$** used for gust analyses (Output)
[QJJL]	Matrix list containing the matrix product: **[TRANS(AJJT)]$^{-1}$** used for nuclear blast analyses (Output)
[AJJDC]	Optional scratch entity to store the intermediate matrix product: **[TRANS(AJJT)]$^{-1}$ * ([D1JK] + ik[D2JK])** from the **QKK** matrix calculation (Output)

Application Calling Sequence:

None

Method:

The **AMP** module begins by querying the **CASE** relation and determining if any **GUST**, **BLAST** and/or **FLUTTER** cases exist. If any of these disciplines or options are selected, the **AMP** module proceeds to compute the requisite matrix lists. The **FLUTTER** bulk data and the **UNMK** data are prepared in core using the **PREFL** and **PRUNMK** utilities. As a separate step, the second record of the **UNMK** is queried to determine the number of aerodynamic interference groups in the model so that the structure of the aerodynamic matrices can be interpreted correctly. As a final initialization task, the existence of both subsonic and supersonic matrices is checked since the **D1JK** and **D2JK** matrices are different for subsonic and supersonic aerodynamics due to the different control point used.

The module then begins to loop through the set of m-k pairs in the **UNMK** entity. For each new Mach number/symmetry group (denoted by the **SGRP** flag), the **UNMK** and **CASE** relation data formed in **PRUNMK** is checked to determine which of the three discipline dependent matrix lists are to be formed for the reduced frequencies associated with the Mach number and symmetry group. If **FLUTTER** or **GUST**

disciplines are associated with the Mach/SGRP set, the corresponding NJ columns of SKJ are extracted from the SKJ list input in the calling sequence. Also, the NJ columns of AJJTL are extracted irrespective of the discipline options. Finally, if the QKK matrix is to be formed, the D1JK and D2JK are processed depending on the presence of both subsonic and supersonic forms. This processing consists of the extraction of the second NK columns of D1JK and D2JK on the first supersonic Mach number encountered. The appropriate matrices are then added together for the current reduced frequency as:

$$[DCJK] = [D1JK] + (0+ik)[D2JK]$$

At this point, the module is ready to deal with the AJJT matrix previously extracted. The processing of this matrix depends on the presence of different interference groups in the unsteady aerodynamics model. For the case with a single interference group, the extracted AJJT matrix is transposed and then decomposed. If the QKK matrix is required, the following matrix is formed using the GFBS utility:

$$[SCRDC] = [AJJ]^{-1}[DCJK]$$

If either the QJJ or QJK matrices are needed, the actual inverse of AJJ is formed and stored as QJJ. If the QJK matrix is needed as well, the QJJ matrix is used to form the QJK matrix as:

$$[QJK] = [SKJ][QJJ]$$

If there is more than one interference group, the alternate path is used to obtain the SCRDC, QJJ and/or QJK matrices. In this path, a loop is performed for each interference group. The second record of the UNMK entity is used to determine the number of j-set and k-set degrees of freedom in the current interference group. These are used to generate the PRTJ partitioning vector for the AJJ matrix. This vector acts as a floating NJG-sized vector to extract the NJG columns and rows associated with the current group. The AJJT matrix is then partitioned, transposed and decomposed to form AJJG. If the QKK matrix is needed, the PRTK partitioning vector is also required. This vector is a floating NKG-sized vector to extract the NKG columns or rows for the current interference group. The DCJK matrix is then partitioned for the current group and used as follows:

$$[AJJDCG] = [AJJG]^{-1}[DCJKG]$$

The INMAT utility is then used to merge this matrix into the SCRDC matrix using the interference group partitioning information. As before, if the QJJ or QJK matrices are needed, the AJJG matrix is inverted and stored as QJJG. The INMAT utility merges this matrix into the QJJ matrix. At the conclusion of the interference group loop, the SCRDC and QJJ matrices are complete. At this point, the logic recombines for both paths. If the QKK matrix is needed, the SCRDC matrix is used to compute QKK as:

$$[QKK] = [SKJ][SCRDC]$$

which is then appended onto the list of QKK matrices, QKKL. If the QJK matrix is needed, the QJJ matrix is used to comput QJK as:

$$[QJK] = [SKJ][QJJ]$$

which is then appended onto the list of QJK matrices, QJKL. Finally, the computed QJJ matrix is appended to the QJJL matrix list if it is required for this m-k/SGRP matrix. The module then continues with the next m-k/SGRP matrix in the UNMK entity. Note that all the matrix lists are formed in the order the m-k/SGRP data appear in the UNMK, although each list need not have all sets. Once the entire set of mk/SGRP sets in the UNMK have been processed, the module terminates by destroying the numerous scratch matrices used in the computations.

Design Requirements:

1. The **FLUTTER** bulk data entries and the **CASE** relation are used to determine the set of m-k/symmetry pairs for each aerodynamic matrices required for each discipline. The data on the database will be used to determine the set of matrices to be computed.

Error Conditions:

None

Engineering Application Module: **ANALINIT**

Entry Point: **ANINIT**

Purpose:

Initializes the final analysis pass. This module should be called at the beginning of the final analysis loop to set parameters as needed for that pass.

MAPOL Calling Sequence:

```
CALL ANALINIT;
```

Application Calling Sequence:

None

Method:

This module is called to perform any actions needed to transition from the optimization segment of ASTROS to the analysis segment. Currently, the only action taken by the module is to overwrite the portion of the **SUBTITLE** that is used to denote the design iteration number (set in **ITERINIT**) with the label **"FINAL ANALYSIS SEGMENT."**

Design Requirements:

1. This routine overwrites the characters 88-128 of the **SUBTIT** variable in **/OUTPT2/** used by **UTPAGE**. No other application modules except **OFP** should modify the **TITLE**, **SUBTIT**, **LABEL** variables beyond the 72nd character, since these fields are used to set dates, page numbers and subcase information.

Error Conditions:

None

Engineering Application Module: **APFLUSH**

Entry Point: **APFLUSH**

Purpose:

Flushs user function related intrinsic response and response gradient entities which are design iteration dependent. This module should be called at the beginning of each design iteration.

MAPOL Calling Sequence:

```
CALL APFLUSH;
```

Application Calling Sequence:

None

Method:

This module must be called at the top of each design iteration loop. It checks the existence of all design iteration dependent relational and matrix entities containing response functions and their gradients, and flushes any that exist.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **AROSNSDR**

Entry Point: **AROSDR**

Purpose:

MAPOL director for saero sensitivity analyses

MAPOL Calling Sequence:

```
CALL AROSNSDR ( NITER, BCID, SUB, NDV, LOOP, MINDEX, CONST, SYM, NGDR,
                [PGDRG(BC)], [UAG(BC)], [AAG(BC)], ACTUAG, [UGA], [AGA], ,
                [PGAA], [PGAU], PCAA, PRAA, [UAGC(BC,SUB)], [AAGC(BC,SUB)],
                ACTAEFF, [AUAGC], [AAAGC], PCAE );
```

NITER	Current iteration number (Input, Integer)
BCID	User defined boundary condition identification number (Integer, Input)
SUB	Current static aeroelastic subscript number (Input, Integer)
NDV	Number of deisgn variables (Input, Integer)
LOOP	A logical flag set to indicate whether additional **MINDEX** subscripts are needed to complete the processing of all the active Mach number/Symmetry conditions on all the **TRIM** entries. One pass for each unique active Mach number will be performed with **MINDEX** set as appropriate for the active pass until this routine returns **LOOP=FALSE** (Logical, Output)
MINDEX	Mach number index value of the current pass (Output, Integer)
CONST	Relation of design constraints (Input)
SYM	Symmetry flag for the current pass. Either 1 for symmetric or -1 for antisymmetric (Output, Integer)
NGDR	Denotes dynamic reduction in the boundary condition (Input, Integer) = **0** No **GDR** = **-1** **GDR** is used
[PGDRG(BC)]	A partitioning vector that removes the additional **GDR** scalar points from the g-set sized displacement and acceleration vectors. Required only if **NGDR** ≠ **0** (Input), where **BC** represents the MAPOL boundary condition loop index number
[UAG(BC)]	g-set displacement vector for all static aero subcases in the current boundary condition (Input), where **BC** represents the MAPOL boundary condition loop index number
[AAG(BC)]	g-set acceleration vector for all static aero subcases in the current boundary condition (Input), where **BC** represents the MAPOL boundary condition loop index number
ACTUAG	Logical flag that is set to **TRUE** if there are any active constraints that require the displacements or accelerations. Those constraints are trim parameters, stresses, strains and displacements (Output, Logical)

[UGA]	Reduced g-set active displacement vectors for all static aero subcases that have active trim parameter, stress, strain and/or displacement constraints. This is a subset of the columns of **[UAG(BC)]** and does not include the **GDR** scalar points, if any (Output)
[AGA]	Reduced g-set active acceleration vectors for all static aero subcases that have active trim parameter, stress, strain and/or displacement constraints. This is a subset of the columns of **[AAG(BC)]** and does not include the **GDR** scalar points, if any (Output)
[PGAA]	Partitioning vector used to obtain **[UGA]** and **[AGA]** from **[UAG(BC)]** and **[AAG(BC)]** (Output)
[PGAU]	Partitioning vector relative to **[UAG(BC)]** and **[AAG(BC)]** that marks the displacement/acceleration columns associated with subcases having active stress, strain or displacement constraints. This vector will be identical to [**PGAA**] unless there are subcases in which **DCONTRM** constraints are active and no stress, strain or displacement constraints are active (Output)
PCAA	An unstructured entity with one word for each active stress, strain or displacement constraint in the current subscript related subcases. That word is the subcase number associated with the constraint (Output)
PRAA	An unstructured entity with one word for each element stress, strain or displacement response function required by the active user function constraints in the current subscript related subcases. That word is the subcase number associated with the response (Character,Output)
[UAGC(BC,SUB)]	g-set pseudodisplacement vectors (displacement fields due to loads arising from unit values of trim configuration parameters) for all aeroelastic effectiveness constraints (Input), where **BC** represents the MAPOL boundary condition loop index number
[AAGC(BC,SUB)]	g-set pseudoacceleration vectors (acceleration fields due to loads arising from unit values of trim configuration parameters) for all aeroelastic effectiveness constraints (Input), where **BC** represents the MAPOL boundary condition loop index number
ACTAEFF	Logical flag that is set to **TRUE** if there are any active constraints that require the pseudodisplacements or pseudoaccelerations. Those constraints are **DCONALE**, **DCONCLA** and **DCONSCF** (Output, Logical)
[AUAGC]	Reduced g-set active pseudodisplacement vectors for all active effectiveness constraints. This is a subset of the columns of **[UAGC(BC)]** and does not include the **GDR** scalar points, if any (Output)
[AAAGC]	Reduced g-set active pseudoacceleration vectors for all active effectiveness constraints. This is a subset of the columns of **[AAGC(BC)]** and does not include the **GDR** scalar points, if any (Output)
PCAE	An unstructured entity with one word for each active effectiveness constraint (**DCONALE**, **DCONCLA**, **DCONSCF**) in the current subscript's related subcases. That word is the column id of the first column associated with the constraint (Output)

Application Calling Sequence:

None

Method:

This module treats two distinct families of aeroelastic constraints for the current boundary condition and subscript number: the active aeroelastic effectiveness constraints **DCONALE**, **DCONCLA** and **DCONSCF**; and the active displacement dependent constraints **DCONTRM**, **DCONDSP**, stress and strain. Two parallel sets of partitioning operations take place to extract the active pseudodisplacements needed for effectiveness constraints and active displacements needed for the displacement-dependent constraints. The control information for the presence or absence of each type of constraint and the additional control information to extract data from downstream entities is also prepared for each constraint family. Finally, the need to loop through another subscript value is determined and the **LOOP** variable is output. **LOOP** will be false after the last needed **AROSNSDR** call for the current **BC**.

First **CASE** is queried to obtain the **TRIM** identification number and symmetry. Then **TRIM** is read to obtain the subscript numbers, **MINDEX** values and subcase ids for each **SAERO** subcase in the current **BC**. These data are then assembled into a master table containing the trim identification number, the subscript number and the subcase id.

The **CONST** relation is then read to count the number of active stress, strain, displacement, aileron effectiveness, lift effectiveness, stability coefficient and trim parameter constraints. A loop over each **CONST** entry is then made to assemble the partitioning vectors and control information for sensitivity computations. Each family of constraints is treated separately.

For effectiveness constraints, the **DISPCOL** attribute in **CONST** is used to build a partitioning vector for the active pseudodisplacements and accelerations. The partitioning vector is later destroyed but the active column numbers are stored as a contiguous string of numbers and written to **PCAE**. For lift effectiveness constraints there is one **UAGC/AAGC** column for each applied constraint: the disp/accel. due to a unit angle of attack. For aileron effectiveness, there are two columns: the first due to unit control surface deflection and the second due to unit roll rate. For stability coefficients, there is one column due to a unit deflection of the constrained parameter. As the constraints are looped over, only those with the current subscript value are considered. Those with lower subscript values have already been processed and, if any active constraints are found with a higher subscript value, the **LOOP** flag is set to **TRUE** to ensure another pass is done.

A similar path exists for the displacement-dependent constraints except the matrices being partitioned are the actual displacement and acceleration fields. Separate partitioning vectors are assembled for 1) active columns due to all displacement dependent constraints (**PGAA**) and 2) active columns due to stress, strain and displacement constraints (**PGUA**). Again, previously processed subscripts are ignored and **LOOP** is set to true if larger subscripts are encountered.

Finally, the assembled partitioning vectors are written to their respective entities and the **PCAE** and **PCAA** entities are determined from the partitioning data and written to the unstructured entities. The presence of active constraints in the effectiveness family or displacement-dependent family is then known and the **ACTAEFF** and **ACTUAG** flags, respectively, are set.

The element stress and strain responses; displacement responses; aeroelastic flexible stability coeffcient responses; and trim parameter responses which are required by active user functional constraints at the current boundary condition and subscript number are treated in the similar manner as those corresponding constraints. The subcases which have active displacement or element stress/strain response functions are also defined as active. The partitioning vector, **PGAA**, and the set of subcase numbers that are active, **PRAA**, are loaded if necessary. The **ACTAEFF** and **ACTUAG** flags are also set for active responses.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **AROSNSMR**

Entry Point: **AROSMR**

Purpose:

Merges the static aero sensitivities for each subscript (stored in the matrix **[MATSUB]**) into the **[MATOUT]** matrix in case order for active subcases rather than subscript order for the current active boundary condition.

MAPOL Calling Sequence:

```
CALL AROSNSMR ( BCID, SUB, NDV, [PGAA], [PGAU], [MATOUT], [MATSUB] );
```

BCID	User defined boundary condition identification number (Integer, Input)
SUB	Current static aeroelastic subscript number (Input, Integer)
NDV	Number of design variables (Input,Integer)
[PGAA]	Partitioning vector used denoting active displacement fields for the current boundary's static aeroelastic subcases (Input)
[PGAU]	Partitioning vector used denoting active displacement fields that are active due only to stress, strain and displacement constraints for the current boundary's static aeroelastic subcases (Input)
[MATOUT]	On input, **MATOUT** must contain the merged, reordered displacement or acceleration sensitivities for all the subcases processed for the earlier subscript values. On output the **SUB**'th subscript is included. This matrix will contain one column for each active vector for the 1st design variable, followed by another set for the second and so on. The order of the vectors within each variable's set will be the order of the **SAERO** subcases in the **CASE** relation (Input and Output)
[MATSUB]	The input matrix of displacement or acceleration sensitivities for all the subcases processed for the **SUB**'th subscript. This matrix will contain one column for each active vector associated with the **SUB**'th subscript for the 1st design variable, followed by another set for the second and so on. The order of the vectors within each variable's set will be the order of the **TRIM** ids appearing in the **TRIM** relation associated with the **SUB**'th subscript value (Input)

Application Calling Sequence:

None

Method:

First the **CASE** relation is read to retrieve the trim id's for the **SAERO** subcases in the current boundary condition. The the **TRIM** relation is read to obtain the subcase numbers associated with each trim id having the current **SUB**script value. Then the **PGAA** and **PGUA** vectors are read into memory to assist in the partitioning operation.

Then the **MATSUB** and **MATOUT** matrices are opened. If **MATOUT** is uninitialized ***or*** if **SUB = 1**, it is initialized (flushed and the number of rows, precision and form set to those of **MATSUB**. If **MATOUT** already exists and has data in it, a scratch matrix is created to hold the final merged data.

For each design variable in the model, each **SAERO CASE** entry for the current boundary is processed. For each **CASE** entry, the partitioning vector **PGAA** is used to determine if it is active and therefore ***may*** have a column in either **MATSUB** or **MATOUT**. For the active subcase id, the **TRIM** data are searched to determine the subscript number associated with the subcase. If the subscript is less than **SUB**, a column from **MATOUT** may be taken (if it was stored there on an earlier pass). If the subscript is equal to **SUB**, it may be stored on the output matrix from **MATSUB**. If greater than **SUB**, it is ignored till later passes.

Once a column is identified as active in **MATSUB** (**PGAA** indicates active and subscript = **SUB**), an additional check is made to see if the column is active in **PGUA**. Only those columns that are active in **PGUA** are copied to **MATOUT**. This filtering is done to limit the amount of computational effort in the stress, strain and displacement constraint sensitivity computations that proceed using the **MATOUT** matrix. The **MATSUB** columns that are active due to **DCONTRM** constraints are no longer needed as these sensitivities are assumed to have been computed already in the **AEROSENS** module.

Once the final matrix is formed, if **MATOUT** had had data in it, the name of the scratch matrix that was loaded is switched with that of **MATOUT**. The scratch entity is then destroyed.

Design Requirements:

1. The assumption is that each **MATSUB** matrix contains the results from the "**SUB**"th subscript value in the order of the trim id's for that **SUB** appear in the **TRIM** relation.

2. The same **MATOUT** matrix must be passed into the **AROSNSMR** module on each call since the columns associated with earlier subscript values are read from **MATOUT** into a scratch entity. The merged matrix that results then replaces the input **MATOUT**.

3. The **AEROSENS** module is called upstream of the **AROSNSMR** module to process active **DCONTRM** constraints for the current subscript. Thus, those columns that are active only for **DCONTRM** constraints may be filtered out for the downstream processing of stress, strain and displacement constraints.

Error Conditions:

None

Engineering Application Module: **BCBGPDT**

Entry Point: **BCBGPD**

Purpose:

Builds the boundary condition-dependent grid point coordinate relation **BGPDT** for the specified boundary condition.

MAPOL Calling Sequence:

```
CALL BCBGPDT ( BCID, GSIZEB, BGPDT(BC), ESIZE(BC) );
```

BCID	User defined boundary condition identification number (Integer, Input)
GSIZEB	Basic g-set size (the size independent of **GDR**-added scalar points) (Integer, Input)
BGPDT(BC)	Relation of basic grid point data for the boundary condition (including any extra points but excluding **GDR** scalar points which may be added by the **GDR** module) (Output), where **BC** represents the MAPOL boundary condition loop index number
ESIZE(BC)	Number of extra point DOF defined for the boundary condition (Integer, Output), where **BC** represents the MAPOL boundary condition loop index number

Application Calling Sequence:

None

Method:

The invarient basic grid point data is read from the **BGPDT** relation (an unsubscripted relation that is formed in **IFP**). The user's extra points selected in the **CASE** relation are then appended in memory and sorted on external id. Uniqueness of the external id's are checked and the new **BGPDT(BC)** is written.

Design Requirements:

1. The invariant **BGPDT** must exist on the data base. It is a hidden output from the **IFP** module.

Error Conditions:

1. Nonunique **GRID/EPOINT** id's are flagged.

Engineering Application Module: **BCBULK**

Entry Point: **BCBULK**

Purpose:

Builds boundary condition-dependent matrices, transfer functions, and initial conditions.

MAPOL Calling Sequence:

```
CALL BCBULK ( BCID, PSIZE(BC), BGPDT(BC), USET(BC) );
```

BCID	User defined boundary condition identification number (Integer, Input)
PSIZE(BC)	The size of the physical set for the current boundary condition (Integer, Input), where **BC** represents the MAPOL boundary condition loop index number
BGPDT(BC)	The relation of basic grid point data for the current **BC** (including any selected extra points) (Input), where **BC** represents the MAPOL boundary condition loop index number
USET(BC)	The unstructured entity of DOF masks for all the points in the current boundary conditions (Input), where **BC** represents the MAPOL boundary condition loop index number

Application Calling Sequence:

None

Method:

All the outputs from this routine are *hidden* — meaning that they do not appear in the call. The purpose of this module is to assemble those data that depend on the boundary condition selection of extra points.

For the data of each type that is referenced in **CASE** for the current boundary condition, the data are retrieved from the bulk data relations that were loaded in **IFP** and are error checked relative to the set of DOF that comprise the current boundary condition. The following hidden entities are output:

BULK DATA	SUBROUTINE	GENERATED ENTITY
DLONLY	PREDOL	UDLOLY
DMIG	PREDMG	named matrix entities
TF	PRETF	TFDATA
IC	PREIC	ICDATA

In each case, these entities contain only those data that relate to the current boundary condition. They will be replaced in subsequent boundary conditions and/or iterations with the appropriate data on each pass.

Design Requirements:

None.

Error Conditions:

1. Initial error checking of each bulk data entry type is performed within this module.

Engineering Application Module: **BCEVAL**

Entry Point: **BCEVAL**

Purpose:

Evaluates the current values of BAR element cross-sectional dimension relationship constraints.

MAPOL Calling Sequence:

```
CALL BCEVAL ( NITER, NDV, GLBDES, LOCLVAR, [PTRANS], CONST );
```

NITER	Design iteration number (Integer,Input)
NDV	Number of design variables (Integer,Input)
GLBDES	Relation of global design variables (Character,Input)
LOCLVAR	Relation containing the relationship between local variables and global variables in the design problem (Character,Input)
[PTRANS]	The design variable linking matrix (Character,Input)
CONST	Relation of constraint values (Character,Output)

Application Calling Sequence:

None

Method:

This module analyzes each designed BAR element which uses PBAR1 property Bulk Data to define its cross-sectional parameters. A set of cross-sectional parameter relationship constraints are computed based on the BAR element cross-sectional shape. Constraints for these relationships are formulated as follows:

"I" Shape:

G1 = (D3 + D4 - D2) / (2*DMAX) and
G2 = (D5 - D1) / DMAX; where **DMAX = MAX(D1,D2,D3,D4,D5)**

"T" Shape:

G1 = (D4 - D1) / DMAX and
G2 = (D3 - D2) / DMAX; where **DMAX = MAX(D1,D2,D3,D4)**

"BOX" Shape:

G1 = (2*D3 - D1) / (2*DMAX) and
G2 = (2*D3 - D2) / (2*DMAX); where **DMAX = MAX(D1,D2,D3)**

"TUBE" Shape:

G1 = (D2 - 0.5*D1) / DMAX; where **DMAX = MAX(D1,D2)**

"HAT" Shape:

G1 = (D3 - D1) / DMAX and
G2 = (2*D4 - D3) / (2*DMAX) and
G3 = (2*D4 + D5 - D2) / (3*DMAX); where **DMAX = MAX(D1,D2,D3,D4,D5)**

"GBOX" Shape:

G1 = (2*D5 + D6 - D1) / (3*DMAX)
G2 = (D3 + D4 - D2) / (2*DMAX); where **DMAX = MAX(D1,D2,D3,D4,D5,D6)**

Note that **D1**, **D2**, **D3**, **D4**, **D5**, and **D6** are **BAR** element cross-sectional parameters.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **BCIDVAL**

Entry Point: **BCIDVL**

Purpose:

Converts the boundary condition index number (**BC**) into the boundary condition identification number (**BCID**).

MAPOL Calling Sequence:

```
CALL BCIDVAL ( BC, CASE, BCID );
```

BC	MAPOL boundary condition loop index number (Integer,Input)
CASE	Relation containing the case parameters for each subcases within each boundary condition (Character,Input)
BCID	User defined boundary condition identification number (Integer, Input)

Application Calling Sequence:

None

Method:

This module counts the boundary contion number through **CASE** entries and obtains the corresponding boundary condition identification number **BCID** from **CASE** for the input boundary condition index number **BC**.

Design Requirements:

None

Error Conditions:

BCID = -1 if relation **CASE** is nonexistent or empty.

Engineering Application Module: **BLASTDRV**

Entry Point: **BLSDRV**

Purpose:

To compute an aircraft's transient response to a nuclear blast.

MAPOL Calling Sequence:

```
CALL BLASTDRV ( BC, [GENM], [GENK], [GENFA], [GENQL], [DELB], [URDB],
                [DWNWSH], [SLPMOD], [ELAS], [UBLASTI] );
```

BC	MAPOL boundary condition loop index number (Integer,Input)
[GENM]	Generalized mass matrix (Input)
[GENK]	Generalized stiffness matrix (Input)
[GENFA]	Mode on box generalized forces (Input)
[GENQL]	Mode on box aeroelastic corrections (Input)
[DELB]	Trim vector of initial conditions (Input)
[URDB]	Vector of initial accelerations (Input)
[DWNWSH]	Matrix of downwash vectors for unit angle of attack and elevator (Input)
[SLPMOD]	Matrix of slopes at box centers due to structural modes (Input)
[ELAS]	Matrix of initial modal displacements (Input)
[UBLASTI]	Matrix of blast response vectors. For each time step, displacement, velocity and acceleration vectors are stored (Output)

Application Calling Sequence:

None

Method:

The module obtains solution control information from the **CASE** relation, the reference chord from **AERO** and blast parameters from **BLAST**. **GENF**, **GENQL**, **DWNWSH**, **SLPMOD** and aerodynamic geometry data are read into core. Initial conditions are set up and a call to subroutine **AST6AS/D** initializes the blast calculation. The **TSTEP** data are then read in and a call to **XBLSTS/D** calculates the blast intercept time. Initial response vectors are written to the **UBLASTI** matrix and a call to subroutine **BLVELS/D** calculates the downwash at the aerodynamic boxes and a call to **AST6AS/D** gets the initial load. A loop on the time steps is then begun. A Newmark-Beta algorithm is applied to calculate the transient response. Within the time loop, calls to **BLVELS/D** and **AST6AS/D** determine the downwash vector and the response, respectively. Data are written to **UBLASTI** as specified by the **TSTEP** input. Once all the time steps have been computed, scratch entities are destroyed and control is returned to the executive.

Design Requirements:

1. This module must come after **BLASTFIT** and **BLASTTRIM**. The module is called in both the optimize and the analyze phase of the MAPOL sequence, but produces no constraint information for the optimization task.

Error Conditions:

None

Engineering Application Module: **BLASTFIT**

Entry Point: **INTRF**

Purpose:

To compute the interpolated time domain steady state and time dependent aerodynamic influence coefficients for blast analyses based on unsteady aerodynamics computed in the time domain.

MAPOL Calling Sequence:

```
CALL BLASTFIT ( BCID, [QJJL], [MATTR], [MATSS], BQDP, [BFRC], [DWNWSH],
                HSIZE(BC), [ID2], [MPART], [UGTKA], [BLGTJA], [BLSTJA] );
```

BCID	User defined boundary condition identification number (Integer, Input)
[QJJL]	Aerodynamic influence coefficient matrices in the frequency domain (Input)
[MATTR]	Aerodynamic influence coefficient matrix list in the time domain (Output)
[MATSS]	Steady stat aerodynamic influence coefficient matrix (Output)
BQDP	Blast dynamic pressure (Real, Output)
[BFRC]	Matrix of aerodynamic forces due to specified rigid body motions (Output)
[DWNWSH]	Matrix of downwash components due to specified rigid body motions (Output)
HSIZE(BC)	Number of normal modes to be used in the analysis (Integer, Output), where **BC** represents the MAPOL boundary condition loop index number
[ID2]	An identity matrix required for performing trim analysis in the MAPOL sequence (Output)
[MPART]	A partitioning vector required for performing trim analysis in the MAPOL sequence (Output)
[UGTKA]	Unsteady spline matrix (Input)
[BLGTJA]	Spline matrix for transforming blast forces (Output)
[BLSTJA]	Spline matrix for transforming displacements and slopes (Output)

Application Calling Sequence:

None

Method:

Problem parameters are first obtained from the **CASE**, **AERO** and **BLAST** entities. A call to **ATMOS** determines the dynamic pressure for the specified flight condition. A list of reduced frequencies that are available for the specified Mach number is retrieved. A grand loop on the number of sending boxes is begun with a call to **DLINDX** to determine the geometry of the sending box. A secondary loop on the receiving box starts with a call to **READDL** to extract the **QJJL** coefficients for the receiving and sending boxes for all the reduced frequencies. These data are then processed using the algorithm described in Appendix B of the Theoretical Manual to compute time domain coefficients. Once all the receiving boxes have been computed, a column of **MATSS** is written and **NBETA** columns are written to scratch matrix **BLSSCR**. Once the loop on sending boxes has been completed, the **BLSSCR** data are retrieved in the order that they are required in **MATTR** and this output matrix is written. A call to **BLSFRS/D** computes matrices

required in the trim analysis. Downwash vectors for given aircraft parameters (such as angle of attack) are computed and stored to **DWNWSH**. Premultiplication of the downwash vector by the **MATSS** matrix computes the forces due to the aircraft parameters and these data are stored in **BLSFRS/D**. Matrices **ID2** and **MPART** are created and the **UGTKA** matrix is partitioned into **BLGTJA** and **BLSTJA**. Control is then returned to the driver subroutine and then to the executive.

Design Requirements:

1. Must follow the **AMP** module. It it is convenient to group this module with the **BLASTTRIM** and **BLASTDRV** modules since they rely on **BLASTFIT** outputs.

Error Conditions:

None

Engineering Application Module: **BLASTRIM**

Entry Point: **BLSTRM**

Purpose:

Performs a trim analysis prior to performing a transient response to a nuclear blast.

MAPOL Calling Sequence:

```
CALL BLASTRIM ( BCID, [DELM], [MRR(BC)], [URDB], [DELB] );
```

BCID	User defined boundary condition identification number (Integer, Input)
BC	Boundary condition index number (Integer, Input)
[DELM]	Matrix of applied loads (Input)
[MRR(BC)]	Reduced mass matrix (Input)
[URDB]	Matrix of aircraft accelerations (Output)
[DELB]	Matrix of trim parameters (Output)

Application Calling Sequence:

None

Method:

The **DELM** and **MRR** matrices are read into core, the **BLAST** identification number is retrieved from the **CASE** entity and the corresponding blast parameters are read from the **BLAST** entity. The trim equations are then solved and the acceleration and trim vectors are written to **URDB** and **DELB**, respectively.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **BOUND**

Entry Point: **BOUND**

Purpose:

To return flags to the MAPOL sequence that define the matrix reduction path for the current boundary condition.

MAPOL Calling Sequence:

```
CALL BOUND ( BCID, GSIZE, ESIZE(BC), USET(BC), BLOAD, BMASS, DMODES, BMODES,
             BSAERO, BFLUTR, BDYN, BDRSP, BDTR, BMTR, BDFR, BMFR, BGUST,
             BBLAST, NMPC, NSPC, NOMIT, NRSET, NGDR );
```

BCID	User defined boundary condition identification number (Integer, Input)
GSIZE	The number of degrees of freedom in the structural set (Integer, Input)
ESIZE(BC)	The number of extra point degrees of freedom in the boundary condition (Integer, Input), where **BC** represents the MAPOL boundary condition loop index number
USET(BC)	The unstructured entity defining structural sets (Character, Input), where **BC** represents the MAPOL boundary condition loop index number
BLOAD	Static load flag; **=1** if any static loads in the current boundary condition (Integer, Output)
BMASS	Mass matrix flag; **=1** if the mass matrix is needed for any discipline(s) in the current boundary condition (Integer, Output)
DMODES	Modes discipline flag; **=1** if any modal dynamic response discipline(s) (Integer, Output)
BMODES	Modal analysis flag; **=1** if any disciplines in the current boundary condition require that a real eigenanalysis be performed (Integer, Output)
BSAERO	Static aeroelastic flag; **=1** if any static aeroelastic analyses are in the current boundary condition (Integer, Output)
BFLUTR	Flutter discipline flag; **=1** if any flutter analyses in the current boundary condition (Integer, Output)
BDYN	Dynamics flag; **=1** if any disciplines requiring dynamic matrix assembly are in the current boundary condition (Integer, Output)
BDRSP	Dynamic response flag; **=1** if any transient or frequency response analyses in the current boundary condition (Integer, Output)
BDTR	Direct Transient Response flag; **=1** if any direct transient response analyses are in the current boundary condition (Integer, Output)
BMTR	Modal Transient Response flag; **=1** if any modal transient response analyses are in the current boundary condition (Integer, Output)
BDFR	Direct Frequency Response flag; **=1** if any direct frequency response analyses are in the current boundary condition (Integer, Output)
BMFR	Modal Frequency Response flag; **=1** if any modal frequency response analyses are in the current boundary condition (Integer, Output)

BGUST	Gust option flag; =1 if any dynamic response disciplines include the **GUST** option in the current boundary condition (Integer, Output)
BBLAST	Blast discipline flag; =1 if any blast analyses are in the current boundary condition (Integer, Output)
NMPC	Number of degrees of freedom in the m-set, (Integer, Output)
NSPC	Number of degrees of freedom in the s-set, (Integer, Output)
NOMIT	Number of degrees of freedom in the o-set, (Integer, Output)
NRSET	Number of degrees of freedom in the r-set, (Integer, Output)
NGDR	Denotes dynamic reduction in the boundary condition. = 0 No **GDR** = -1 **GDR** is used (Output, Integer)

Application Calling Sequence:

None

Method:

The **USET** entity and **CASE** relation are read to determine the sizes of the dependent structural sets and to ensure that no illegal combinations of disciplines and matrix reduction methods reside in the same boundary condition. The matrix reductions and analysis steps in the standard MAPOL sequence are then guided by the flags from **BOUND**. A summary of the structural sets is printed to the output file followed by a summary of the disciplines and subcases that have been selected.

Design Requirements:

1. The **CASE** relation must be filled with the information from the Solution Control Packet by the **SOLUTION** module. Also, the **MKUSET** module must have loaded the **USET** entity.

Error Conditions:

None

Engineering Application Module: **BOUNDUPD**

Entry Point: **BOUNDUPD**

Purpose:

To echo the updated boundary condition summary.

MAPOL Calling Sequence:

```
CALL BOUNDUPD ( BCID, GSIZE, ESIZE(BC), USET(BC),NSPC, NOMIT, NRSET );
```

BCID	User defined boundary condition identification number (Integer, Input)
GSIZE	The number of degrees of freedom in the structural set (Integer, Input)
ESIZE(BC)	The number of extra point degrees of freedom in the boundary condition (Integer, Input), where **BC** represents the MAPOL boundary condition loop index number.
USET(BC)	The unstructured entity defining structural sets (Character, Input), where **BC** represents the MAPOL boundary condition loop index number.
NSPC	Number of degrees of freedom in the s-set, (Integer, Output)
NOMIT	Number of degrees of freedom in the o-set, (Integer, Output)
NRSET	Number of degrees of freedom in the r-set, (Integer, Output)

Application Calling Sequence:

None

Method:

The **USET** entity and **CASE** relation are read to determine the sizes of the dependent structural sets and to ensure that no illegal combinations of disciplines and matrix reduction methods reside in the same boundary condition. The matrix reductions and analysis steps in the standard MAPOL sequence are then guided by the flags from **BOUND**. A summary of the structural sets is printed to the output file followed by a summary of the disciplines and subcases that have been selected.

Design Requirements:

1. The **CASE** relation must be filled with the information from the Solution Control Packet by the **SOLUTION** module. Also, the **MKUSET** module must have loaded the **USET** entity.

Error Conditions: **None**

Engineering Application Module: **CONORDER**

Entry Point: **CONORD**

Purpose:

Reorders active constraints in boundary condition order to match the order in which constraint sensitivities are computed.

MAPOL Calling Sequence:

```
CALL CONORDER ( NITER, NUMOPTBC, CASE, CONST, CONSTORD );
```

NITER	Design iteration number (Integer,Input)
NUMOPTBC	Number of optimization boundary conditions (Integer,Input)
CASE	Relation containing the case parameters for each subcases within each boundary condition (Character,Input)
CONST	Relation of constraint values (Character,Input)
CONSTORD	Relation of reordered constraint values (Character,Output)

Application Calling Sequence:

None

Method:

This module first gets boundary condition independent active constraints (thickness constraints, cross-sectional parameter constraints for BAR's, and laminate gauge constraints) from relation **CONST** and put them into the new relation **CONSTORD**. Then, for each optimization boundary condition, active frequency and flutter constraints are boundary condition dependent, but not subcase dependent, therefore they precede any subcase dependent constraints for the current boundary condition in relation **CONSTORD**. Thereafter: active STATICS displacement and stress/strain constraints are placed in subcase order; active SAERO aeroelastic effectiveness and stability coefficient constraints, trim parameter constraints are placed in subscript order; and active SAERO displacement and stress/strain constraints are placed in subcase order. Finally, active panel buckling constraints and Euler buckling constraints are placed in subcase order in relation **CONSTORD**.

Design Requirements:

Since the reordered constraint relation **CONSTORD** is required by module **OFPGRAD** and **DESIGN**, module **CONORDER** must be called prior to those modules and after module **ACTCON**. Module **CONORDER** must be placed at the outside of optimization boundary condition loop.

Error Conditions:

None

Engineering Application Module: **DCEVAL**

Entry Point: **DCEVAL**

Purpose:

To evaluate displacement constraints in the current boundary condition.

MAPOL Calling Sequence:

```
CALL DCEVAL ( NITER, BCID, [UG(BC)], BGPDT(BC), CONST, BSAERO );
```

NITER	Design iteration number (Integer, Input)
BCID	User defined boundary condition identification number (Integer, Input)
[UG(BC)]	Matrix of displacement vectors in the g-set for the boundary condition (Input), where **BC** represents the MAPOL boundary condition loop index number.
BGPDT(BC)	Relation of basic grid point coordinate data (Character, Input), where **BC** represents the MAPOL boundary condition loop index number.
CONST	Relation of constraint values (Character, Input)
BSAERO	Static aeroelastic flag; **=1** if this call is associated with static aeroelastic analyses. (Optional, Integer, Input)

Application Calling Sequence:

None

Method:

The module first determines if there are any **DCONST** options for a **STATIC** (**BSAERO=0**) or **SAERO** (**BSAERO=1**) discipline for the current boundary condition and terminates if there are none. If there are, a loop is made through all the subcases for the current boundary condition and the necessary displacement constraint(s) are calculated and written to the **CONST** relation. Finally, the displacement responses which are required by any user functional constraints are computed.

Design Requirements:

1. This module appears within the analysis portion of the **OPTIMIZE** segment of the MAPOL sequence. It is within the analysis boundary condition loop and must follow the recovery of the displacement vector to the g-set.

Error Conditions:

None

Engineering Application Module: **DDLOAD**

Entry Point: **DDLOAD**

Purpose:

To compute the sensitivities of design dependent loads for active boundary conditions.

MAPOL Calling Sequence:

```
CALL DDLOAD ( NDV, GSIZEB, BCID, SMPLOD, [DPTHVI], [DPGRVI], [DDPTHV],
              [DDPGRV], DDFLG, [PGAS], [DPVJ] );
```

NDV	The number of global design variables (Integer, Input)
GSIZEB	The size of the structural set (Integer, Input)
BCID	User defined boundary condition identification number (Integer, Input)
SMPLOD	Unstructured entity of simple load vector information (Input)
[DPTHVI]	Matrix entity containing the linearly designed thermal loads (Character, Input)
[DPGRVI]	Matrix entity containing the linearly designed gravity loads (Character, Input)
[DDPTHV]	Matrix entity containing the nonlinear thermal load sensitivity (Character, Input)
[DDPGRV]	Matrix entity containing the nonlinear gravity load sensitivity (Character, Input)
DDFLG	Design dependent load flag: (Integer, Output) = **0** if no design dependent loads = **1** if any static loads are design dependent
[PGAS]	Matrix entity containing a partitioning vector of active applied static load conditions (Input)
[DPVJ]	Matrix entity containing the senstivities of each active static load to the design variables (Output)

Application Calling Sequence:

None

Method:

The module first determines if there are any static loads and if any of the applied static loads are potentially design dependent. This is done by reading the **SMPLOD** entity and checking if any gravity or thermal loads are defined. If any design dependent applied loads are found, the module continues by reading the remainder of the first **SMPLOD** record, the **CASE** relation for all **STATICS** disciplines in the current active boundary condition and all the **LOAD** relational tuples. Finally, the **PGA** vector is brought into core to allow the active loads to be identified. Once all the data are in core, the **PGA** data are used to identify the active static loads. For each active load, the **CASE** relation is searched to determine if any of the simple loads comprising the current active load are design dependent. This involves the **LOAD** relational data for **MECH** loads since the **LOAD** data may refer to **GRAV** loads which are design dependent. If any design dependent loads are found, their sensitivities are computed using the **DPGRVI**, **DDPGRV**, **DPTHVI** and/or **DDPTHV** matrix entities of simple load sensitivities. The **DPVJ** entity is loaded as active design

dependent loads are encountered with care taken that ***all*** active loads (including design independent loads) are accounted for in the column dimension of the matrix entity.

Design Requirements:

1. This module must be called to initialize the **DDFLG** flag that is used by the MAPOL sequence to direct subsequent matrix operations relating to the load sensitivities even if no design dependent loads exist in the boundary condition.

2. The module assumes that at least one active static applied load exists in the current boundary condition.

Error Conditions:

None

Engineering Application Module: **DESIGN**

Entry Point: **DESIGN**

Purpose:

To perform redesign by math programming methods based on the current set of active constraints and constraint sensitivities.

MAPOL Calling Sequence:

```
CALL DESIGN ( NITER, NDV, APPCNVRG, MOVLIM, CNVRGLIM, CTL, CTLMIN,
              GLBDES, CONST, CONSTORD, [AMAT], DESHIST );
```

NITER	Design iteration number (Integer, Input)
NDV	The number of design variables (Integer, Input)
APPCNVRG	The approximate problem converge flag (Logical, Output) = **FALSE** if not converged = **TRUE** if converged in objective function value
MOVLIM	Limit on how much a design variable can move for this iteration (Real, Input)
CNVRGLIM	Tolerance for indicating approximate problem convergence (Real, Input)
CTL	Tolerance for indicating an active constraint (Real, Output)
CTLMIN	Tolerance for indicating a violated constraint (Real, Output)
GLBDES	Relation of global design variables (Character, Input)
CONST	Relation of constraint values (Character, Input)
CONSTORD	Relation of reordered constraint values (Character, Input)
[AMAT]	Matrix of constraint sensitivities (Input)
DESHIST	Relation of design iteration information (Character, Output)

Application Calling Sequence:

None

Method:

The module first brings design variable, objective, constraint, objective sensitivity and constraint sensitivity information into core. Calls to **ADS** then invoke the mathmatical programming algorithm which performs the redesign task. Function evaluations and gradient evaluations that are required as part of the math programming task are performed by subroutines **FEVAL** and **GREVAL**, respectively.

For a user defined objective function and user function constraints, all required response functions and their sensitivities are computed prior to this module. This module computes user function values and sensitivities by calling the user function evaluation utilities in subroutines **FEVAL** and **GREVAL**, respectively.

Once the appoximate optimizaton process is complete, the **GLBDES** relation is updated with the new values of the design variables and a new entry is written to the **DESHIST** relation.

Design Requirements:

1. This module is called after all the analysis and gradient information has been computed for a design iteration. It is therefore the last module within the design iteration loop.

Error Conditions:

1. The module does not have sufficient memory available.

Engineering Application Module: **DESPUNCH**

Entry Point: **DESPCH**

Purpose:

Punchs out the new Bulk Data cards with property values representing the current design model. The activation of this module depends on the **PUNCH** input.

MAPOL Calling Sequence:

```
CALL DESPUNCH ( NITER, PUNCH, OLOCALDV ) ;
```

NITER	Current design iteration number (Integer, Input)
PUNCH	Logical flag indicating that the user has requested the new model be punched (Logical, Input)
OLOCALDV	Relation of current local design variable values (Input)

Application Calling Sequence:

None

Method:

This module works in conjunction with the **IFP** module and the **ACTCON** module. **IFP** stores the design invariant bulk data and the design variant bulk data as a series of data base entities that are read here. The **ACTCON** module actually computes the current local design variable values and loads them in the **OLOCALDV** relation. It also sets the **PUNCH** logical flag if punched output has been requested for the current design iteration.

If the **PUNCH** flag is true, the data in the **OLOCALDV** relation are looped over and the new property and/or connectivity bulk data entries are punched to the ASTROS punch file.

Design Requirements:

1. **IFP** must have been called to store the bulk data deck for punch requests.
2. **ACTCON** must have been called during the current iteration to load the **OLOCALDV** relation.

Error Conditions:

None

Engineering Application Module: **DMA**

Entry Point: **DMA**

Purpose:

To assemble the direct and/or modal stiffness, mass and/or damping matrices including extra point degrees of freedom for transient, frequency and blast disciplines.

MAPOL Calling Sequence:

```
CALL DMA ( NITER, BCID, ESIZE(BC), PSIZE(BC), BGPDT(BC), USET(BC), [MAA],
           [KAA], [TMN(BC)], [GSUBO(BC)], NGDR, LAMBDA, [PHIA], [MDD], [BDD],
           [KDDT], [KDDF], [MHH], [BHH], [KHHT], [KHHF]);
```

NITER	Design iteration number (Integer, Input)
BCID	User defined boundary condition identification number (Integer, Input)
ESIZE(BC)	The number of extra point degrees of freedom in the boundary condition (Integer, Input), where **BC** represents the MAPOL boundary condition loop index number.
PSIZE(BC)	The size of the physical set for the current boundary condition. (Integer, Input), where **BC** represents the MAPOL boundary condition loop index number.
BGPDT(BC)	Relation of basic grid point coordinate data (Character, Input), where **BC** represents the MAPOL boundary condition loop index number.
USET(BC)	The unstructured entity defining structural sets (Character, Input), where **BC** represents the MAPOL boundary condition loop index number.
[MAA]	Matrix entity containing the mass matrix in the analysis set (Input)
[KAA]	Matrix entity containing the stiffness matrix in the analysis set (Input)
[TMN(BC)]	Transformation matrix for multipoint constraints (Input), where **BC** represents the MAPOL boundary condition loop index number.
[GSUBO(BC)]	Transformation matrix for reduction to the analysis set (Input), where **BC** represents the MAPOL boundary condition loop index number.
NGDR	Denotes dynamic reduction in the boundary condition. (Input, Integer) =0 No **GDR** = –1 **GDR** is used
LAMBDA	Relational entity containing the output from the real eigenanalysis (Input)
[PHIA]	Matrix of eigenvectors from the real eigenanalysis in the analysis set (Input)
[MDD]	Direct dynamic mass matrix (Output)
[BDD]	Direct dynamic damping matrix (Output)
[KDDT]	Direct transient stiffness matrix (Output)
[KDDF]	Direct frequency stiffness matrix (Output)
[MHH]	Modal dynamic mass matrix (Output)

[BHH]	Modal dynamic damping matrix (Output)
[KHHT]	Modal transient stiffness matrix (Output)
[KHHF]	Modal frequency/flutter stiffness matrix (Output)

Application Calling Sequence:

None

Method:

The module begins by retrieving all the **CASE** tuples for **TRANSIENT**, **FREQUENCY** or **BLAST** disciplines for the current boundary condition. If any dynamic matrix assembly is required, the **BGPDT** data is also retrieved and the number of extra points in the current boundary condition is determined and the **PSIZE** variable set to be the size of the physical set. Continuing with the module initialization, the **DMAPVC** submodule is called to generate all the partitioning vectors for the dynamic degrees of freedom including the extra points. If there are extra points, the module proceeds to expand the analysis set structural matrices, mode shapes and transformation matrices to include the extra point degrees of freedom. This is done in the **DMAEXP** submodule.

Next, the **DMAX2** submodule is called to assemble any direct matrix input. These include **M2PP**, **B2PP** and **K2PP** inputs as well as transfer function data. The **DMAX2** submodule forms the zeroth, first and second order inputs in the direct dynamic degrees of freedom. The modal form is obtained during the actual dynamic matrix assembly.

The module then proceeds to obtain the information needed to assemble the damping matrix. The **DAMPING** attribute of the **CASE** relation is checked and the **VSDAMP** and **TABDMP** entries are searched for a matching identification number. Logical flags are set to indicate that damping, modal damping and/or direct damping are to be used. If modal damping is selected, the **LAMBDA** relation is read to obtain the natural frequencies for the computed modes. As a final initialization task, the **DMA** module prepares the data needed to generate the d-sized hidden matrix entity **ICMATRIX** used to perform direct transient analysis. The **ICDATA** information is brought into open core and the p-sized scratch matrix to be reduced to **ICMATRIX** is created.

Once these initialization tasks have been completed, the loop to form the direct and/or modal dynamic matrices begins. The **CASE** relation tuples for the dynamic disciplines are searched sequentially and the requisite matrices formed. Note that the restrictions in the definition of a boundary condition make it such that only one form of each matrix is possible. The **DMA** module forms up to eight matrices: **[MDD]**, **[BDD]**, **[KDDF]**, **[MHH]**, **[BHH]**, **[KDDT]**, **[KHHF]**, and **[KHHT]** depending on the requested disciplines and discipline options. The **INFO** arrays for the matrices are used to store flags denoting coupled or uncoupled matrices and the form of the damping used in the modal stiffness and/or damping matrices. When all the **CASE** tuples have been searched and the required dynamic matrices formed, the module begins the cleanup. The first task is to complete the generation of the initial conditions matrix by reducing the scratch p-sized matrix to the direct dynamic set. Following this action, the other scratch matrices used in the module are destroyed and control returned to the executive.

Design Requirements:

1. The **PFBULK** module must have been called to perform the preprocessing for the initial conditions, transfer functions, and direct matrix input.

Error Conditions:

None

Engineering Application Module: **DYNLOAD**

Entry Point: **DYNLOD**

Purpose:

To assemble the direct and/or modal time and/or frequency dependent loads including extra point degrees of freedom for dynamic response disciplines.

MAPOL Calling Sequence:

```
CALL DYNLOAD ( NITER, BCID, GSIZE, ESIZE(BC), PSIZE(BC), SMPLOD, BGPDT(BC),
               USET(BC), [TMN(BC)], [GSUBO(BC)], NGDR, [PHIA], [QHJL], [PDT],
               [PDF], [PTGLOAD], [PTHLOAD], [PFGLOAD], [PFHLOAD] );
```

NITER	Design iteration number (Integer, Input)
BCID	User defined boundary condition identification number (Integer, Input)
GSIZE	Length of the g-set vector (Integer, Input)
ESIZE(BC)	The number of extra point degrees of freedom in the boundary condition (Integer, Input), where **BC** represents the MAPOL boundary condition loop index number.
PSIZE(BC)	The size of the physical set for the current boundary condition. (Integer, Input), where **BC** represents the MAPOL boundary condition loop index number.
SMPLOD	Unstructured entity of simple load vector information (Input)
BGPDT(BC)	Relation of basic grid point coordinate data (Input), where **BC** represents the MAPOL boundary condition loop index number.
USET(BC)	The unstructured entity defining structural sets (Input), where **BC** represents the MAPOL boundary condition loop index number.
[TMN(BC)]	Matrix for reducing **MPC**s (Input), where **BC** represents the MAPOL boundary condition loop index number.
[GSUBO(BC)]	Matrix for reducing omitted DOF (Input), where **BC** represents the MAPOL boundary condition loop index number.
NGDR	Denotes dynamic reduction in the boundary condition. =0 No **GDR** = –1 **GDR** is used (Input, Integer)
[PHIA]	Natural modes matrix in the a-set (Input)
[QHJL]	Aerodynamic matrix for gust (Input)
[PDT]	Dynamic load vector for transient analysis (Input)
[PDF]	Dynamic load vector for frequency analysis (Input)
[PTGLOAD]	Applied load matrix for the time dependent loads when **LOAD** print is requested (Character, Output)
[PTHLOAD]	Applied load matrix for the time dependent loads when **MODAL GUST** print is requested (Character, Output)

[PFGLOAD]	Applied load matrix for the frequency dependent loads when **LOAD** print is requested (Character, Output)
[PFHLOAD]	Applied load matrix for the frequency dependent loads when **MODAL GUST** print is requested (Character, Output)

Application Calling Sequence:

None

Method:

The module first interrogates the **CASE** relation to see whether any dynamic analyses are to be performed for the current boundary condition. If not, the module terminates. Error checking is performed to make sure legal requests have been made and bookkeeping is performed to set up for matrix reductions and extra points. Call(s) are then made to **DMAPG** to generate the applied loads in the p-set. **DMAPG** reduces these loads to the d-or h-set, depending on the approach. Separate routines generate loads in the frequency and time domains.

Design Requirements:

1. Follows computation of quantities in the a-set. If the **MODAL** approach is being used, the natural mode shapes must be computed.

Error Conditions:

1. No more than one frequency and/or transient load is allowed per boundary condition.

Engineering Application Module: **DYNRSP**

Entry Point: **DYNRSP**

Purpose:

To compute the direct or modal displacements, velocities and accelerations for transient and frequency analyses.

MAPOL Calling Sequence:

```
CALL DYNRSP ( BCID, ESIZE(BC), [MDD], [BDD], [KDDT], [KDDF], [MHH], [BHH],
              [KHHT], [KHHF], [PDT], [PDF], [QHHL], [UTRANA], [UFREQA],
              [UTRANI], [UFREQI], [UTRANE], [UFREQE] );
```

BCID	User defined boundary condition identification number (Integer, Input)
ESIZE(BC)	The number of extra point degrees of freedom in the boundary condition (Integer, Input), where **BC** represents the MAPOL boundary condition loop index number.
[MDD]	Mass matrix in the d-set (Input)
[BDD]	Damping matrix in the d-set (Input)
[KDDT]	Stiffness matrix in the d-set for transient analyses (Input)
[KDDF]	Stiffness matrix in the d-set for frequency analyses (Input)
[MHH]	Modal mass matrix (Input)
[BHH]	Modal damping matrix (Input)
[KHHT]	Modal stiffness matrix for transient analyses (Input)
[KHHF]	Modal stiffness matrix for frequency analyses (Input)
[PDT]	Matrix of applied loads for transient analysis (Input)
[PDF]	Matrix of applied loads for frequency analysis (Input)
[QHHL]	Generalized aerodynamic forces for gust analyses (Input)
[UTRANA]	Transient response vectors in the a-set (Output)
[UFREQA]	Frequency response vectors in the a-set (Output)
[UTRANI]	Transient response vectors in the i-set (Output)
[UFREQI]	Frequency response vectors in the i-set (Output)
[UTRANE]	Transient response vectors in the e-set (Output)
[UFREQE]	Frequency response vectors in the e-set (Output)

Application Calling Sequence:

None

Method:

The module first interrogates the **CASE** relation to see whether any dynamic analyses are to be performed for the current boundary condition. If not, the module terminates. Bookkeeping is performed to set up for any gust analyses and to process extra points. A loop on the number of cases with dynamic response requirements for the current boundary condition is then made. Time or frequency points at which the response is required are established and the required analyses are performed. Separate subroutines control the performance of requested analyses:

ROUTINES	PURPOSE
TRUNCS/D	Uncoupled transient analysis
TRCOUP	Coupled transient analysis
FRUNCS/D	Uncoupled frequency analysis
FRCOUP	Coupled frequency analysis
FRGUST	Frequency response with gust

These routines fill output vectors with response quantities (displacement, velocity and acceleration). If there are extra points, a partitioning operation is performed to segregate extra point data into separate matrix entities.

Design Requirements:

1. Modules **DMA** and **DYNLOAD** prepare matrix quantities that are required for this module. If a gust analysis is being performed, module **QHHLGEN** must have been processed as well.

Error Conditions:

None

Engineering Application Module: **EBKLEVAL**

Entry Point: **EBKEVA**

Purpose:

Evaluates the current values of the Euler buckling constraints.

MAPOL Calling Sequence:

```
CALL EBKLEVAL ( BCID, NITER, NDV, GLBDES, LOCLVAR, [PTRANS], CONST,
                FDSTEP, OEULBUCK );
```

BCID	User defined boundary condition identification number (Integer, Input)
NITER	Design iteration number (Integer,Input)
NDV	Number of design variables (Integer,Input)
GLBDES	Relation of global design variables (Character,Input)
LOCLVAR	Relation containing the relationship between local variables and global variables in the design problem (Character,Input)
[PTRANS]	The design variable linking matrix (Character,Input)
CONST	Relation of constraint values (Character,Output)
FDSTEP	Relative design variable increment for finite difference computation (Real,Input)
OEULBUCK	Relation containing Euler buckling constraint output (Character,Output)

Application Calling Sequence:

None

Method:

This module first checks if any **DCONBKE** Bulk Data entries are referenced by any STATICS and/or SAERO disciplines for the current boundary condition to determine if there any Euler buckling constraints have been applied. If any are found, the BAR and/or ROD element data are obtained from relation **BEAMEST** and/or **RODEST**, and the element force data are obtained from relation **EOBAR** and/or **EOROD**, and the Euler buckling constraint values are evaluated and stored into relation **CONST**. The constraint sensitivity data are also prepared in this module.

Design Requirements:

This module needs element output relation from module **EDR**, therefore should only be called after module **EDR**.

Error Conditions:

1. Euler buckling control elements with improper boundary condition type are flagged.
2. Euler buckling control elements with improper moment of inertia parameters are flagged.

Engineering Application Module: **EBKLSENS**

Entry Point: **EBKSNS**

Purpose:

Evaluates the Euler buckling constraint sensitivity.

MAPOL Calling Sequence:

```
CALL EBKLSENS ( BCID, NITER, NDV, CONST, DESLINK, GLBDES, [AMAT] );
```

BCID	User defined boundary condition identification number (Integer, Input)
NITER	Design iteration number (Integer,Input)
NDV	Number of design variables (Integer,Input)
CONST	Relation of constraint values (Character,Input)
DESLINK	Relation of design variable linking information (Character,Input)
GLBDES	Relation of global design variables (Character,Input)
[AMAT]	Matrix containing the sensitivity of the constraints to changes in the design variable (Character,Output)

Application Calling Sequence:

None

Method:

This module first checks if there any active Euler buckling constraints for the current boundary condition. If so, the constraint sensitivity data are retrieved from relation **CONST**. Then the design variable identification list is then obtained from relation **GLBDES**, and the design variable linking data are obtained from relation **DESLINK**. For each active Euler buckling constraint, the sensitivity to the design variables is computed and stored into matrix **[AMAT]**.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **EDR**

Entry Point: **EDRDRV**

Purpose:

To compute the stresses, strains, grid point forces and strain energies for elements selected for output for the particular boundary condition.

MAPOL Calling Sequence:

```
CALL EDR ( BCID, NITER, NDV, GSIZE, EOSUMMRY, EODISC, GLBDES,
           LOCLVAR, [PTRANS], [UG(BC)], [UAG(BC)], [BLUG], [UTRANG],
           [UFREQG], [PHIG(BC)], [PHIGB(BC)] );
```

BCID	User defined boundary condition identification number (Integer, Input)
NITER	Iteration number for the current design iteration (Integer, Input)
NDV	The number of global design variables (Integer, Input)
GSIZE	The size of the structural set (Integer, Input)
EOSUMMRY	Relation containing the summary of elements, design iterations and boundary conditions for which output is desired (Input)
EODISC	Unstructured entity referred to by **EOSUMMRY** containing the disciplines for which output is required for each element/iteration/boundary condition (Input)
GLBDES	Relation of global design variables (Character, Input)
LOCLVAR	Relation containing the relationship between local variables and global variables in the design problem (Character, Input)
[PTRANS]	The design variable linking matrix (Character, Input)
[UG(BC)]	Matrix of global displacements from **STATICS** analyses (Input), where **BC** represents the MAPOL boundary condition loop index number.
[UAG(BC)]	Matrix of global displacements from **SAERO** analyses (Input), where **BC** represents the MAPOL boundary condition loop index number.
[BLUG]	Matrix of global displacements/velocities/accelerations for **BLAST** response analyses (Input)
[UTRANG]	Matrix of global displacements/velocities/accelerations for **TRANSIENT** response analyses (Input)
[UFREQG]	Matrix of global displacements/velocities/accelerations for **FREQUENCY** response analyses (Input)
[PHIG(BC)]	Matrix of global eigenvectors from real eigenanalysis for **MODES** analyses (Input), where **BC** represents the MAPOL boundary condition loop index number.
[PHIGB(BC)]	Matrix of global eigenvectors for **BUCKLING** analyses (Input), where **BC** represents the MAPOL boundary condition loop index number.

Application Calling Sequence:

None

Method:

The **EOSUMMRY** relation is opened and read for the current boundary condition. If any element output requests exist, processing continues by loading the input matrices associated with the discipline dependent displacement fields into an character array such that the order in which disciplines are processed correspond to the order of the matrices. Following this, there is a section of code set aside for discipline dependent processing. Currently, two tasks are performed:

(1) The number of mode shapes computed in the real eigenanalysis (if one was performed) is determined by opening the **PHIG** matrix; and (2) any thermal load set ID's in the **CASE** relation are replaced by the record number in **GRIDTEMP** that corresponds to the applied load case.

The initialization tasks continue with a call to the **PRELDV** utility to set up for computation of the local design variables associated with designed elements. The transformation matrices and material properties are also prepared for fast retrieval by the element routines. The **GPFDATA** relation is opened for output and the **EODISC** data is read into memory. At this point, the **EODISC** record number in the **EOSUMMRY** data is replaced by the open core pointer where the record begins in memory. With the initialization complete, the **EDR** module proceeds to compute the desired element response quantities for all the "subcases" (considered by **EDR** to be represented by a single displacement vector) for any or all disciplines that have been analyzed in the current boundary condition. The computation occurs in the following three steps:

(1) Determine the set of disciplines and subcases for which any element response quantities are needed

(2) Read into open core as many displacement vectors (real and/or complex) as will fit

(3) Call element dependent routines to compute the stress, strain, strain energy, forces and grid point forces for each displacement vector

To perform step (1), the **EOSUMMRY** data is read for each discipline and the corresponding **EODISC** data is used to form a unique list of subcases for each discipline in the current boundary condition. A list of the form:

```
NDISC,(DISC TYPE(I),NSUBCASE,SUBCASE ID(J),J=1,NSUBCASE),I=1,NDISC)
```

These data are sorted by discipline type in the order defined in the **/EDRDIS/** common block and by the subcase "identification numbers" within each discipline. The subcase ID's refer to the column number in the displacement matrix for the discipline. For statics and modes, these numbers are incremented by one for each new load condition or eigenvector while transient, frequency and blast results use an increment of three to accommodate the velocity and accelerations that are stored in the same matrix. After this in-core list has been formed, it is read to determine which displacement vectors are to be brought into open core. The module determines the amount of remaining memory and brings as many displacement vectors into memory as possible. The terms are converted to single precision at this point. Once all the displacements are in memory, or memory is full, the element dependent routines are called. Within each element dependent routine, the geometrical portion of the element processing is performed once followed by a loop over all incore displacements to compute the element response quantities. For each displacement set, all the element response quantities including grid point forces, stresses, strains, strain energies and element forces are computed and stored on the **EO**xxxx element response quantity relations. Note that the exact quantities requested by the user are not used at this point, but will only be used to determine which data to print. Once all the elements have been processed, the module loops back for any remaining displacement vectors and, when all of these are processed, terminates.

Design Requirements:

1. The **PFBULK** processing of the element output requests must have been completed and be compatible with the data currently resident in the **CASE** relation.

2. The module may be called when no element output requests exist in the Solution Control.

Error Conditions:

None

Engineering Application Module: **EMA1**

Entry Point: **EMA1**

Purpose:

To assemble the linearly designed element stiffness and mass matrices (stored in the **KELM** and **MELM** entities) into the linear design sensitivity matrices **DKVI0** and **DMVI0**.

MAPOL Calling Sequence:

```
CALL EMA1 ( NDV, CSTM, GENEL, DVCT, KELM, MELM, GMKCT0, DKVI0, GMMCT0,
            DMVI0, DWGH1 );
```

NDV	Number of design variables (Integer, Input)
CSTM	Relation containing the coordinate transformation matrices for all external coordinate systems (Character,Input)
GENEL	Unstructured entity containing information from **GENEL** Bulk Data entries (Character,Input)
DVCT	Relation containing the data required for the assembly of the linear design sensitivity matrices (Character, Input)
KELM	Unstructured entity containing the linear design element stiffness matrix partitions (Character, Input)
MELM	Unstructured entity containing the linear design element mass matrix partitions (Character, Input)
GMKCT0	Relation containing connectivity data for the **DKVI0** sensitivity matrix (Output)
DKVI0	Unstructured entity containing the linear design stiffness sensitivity matrix in a highly compressed format (Output)
GMMCT0	Relation containing connectivity data for the **DMVI0** sensitivity matrix (Output)
DMVI0	Unstructured entity containing the linear design mass sensitivity matrix in a highly compressed format (Output)
DWGH1	Unstructured entity containing the linear (invariant) part of the sensitivity of weight to the design variables (Output)

Application Calling Sequence:

None

Method:

This module deals with linearly designed stiffness and mass matrices, while module **NLEMA1** deals with nonlinear design stiffness and mass matrices. The module is executed in two passes; once for linear design stiffness matrices and a second time for linear design mass matrices. In the first pass, **DVCT** information is read into core one record at a time. The algorithm is structured to maximize the amount of processing done on a given design sensitivity matrix (typically all of it) in core. Spill logic is in place if a matrix cannot be completely held in core. For the assembly, subroutine **RQCOR1** performs bookkeeping tasks to expedite the assembly and to determine whether spill will be necessary. Subroutine **ASSEM1** retrieves **KELM** information, performs the actual assembly operations and places the results into the **GMKCT0** and **DKVI0** entities. When the **DVCT** data have been exhausted a check is made as to whether

mass assembly is required. If a discipline which requires a mass matrix is included in the solution control, the mass terms are assembled in the second pass. If there are **OPTIMIZE** boundary conditions, this module calculates the linear portion of sensitivity of the objective to the design variables regardless of whether the **DMVI0** matrices are required. If no mass information is required, control is returned to the executive and the second pass through the module is not made. For the second pass, **MELM** data are used. The structure of the assembly operation is otherwise much the same and **GMMCT0** and **DMVI0** data are computed and stored.

Design Requirements:

1. This assembly operation follows the **MAKEST** and **EMG** modules.

2. Since gravity loads require **DMVI0** data, it is necessary to perform **EMA1** prior to calling **LODGEN**. **EMA1** must always be called before **EMA2**.

Error Conditions:

None

Engineering Application Module: **EMA2**

Entry Point: **EMA2**

Purpose:

To assemble the element stiffness and mass matrix partitions (stored in the **DKVIG** and **DMVIG** entities) into the global stiffness and mass matrices for the current design iteration.

MAPOL Calling Sequence:

```
CALL EMA2 ( NITER, NDV, GSIZEB, GLBDES, GMKCTG, DKVIG, [K1GG], GMMCTG,
            DMVIG, [M1GG] );
```

NITER	Design iteration number (Integer, Input)
NDV	The number of design variables (Integer, Input)
GSIZEB	Length of the g-set vectors (Integer, Input)
GLBDES	Relation of global design variables (Character, Input)
GMKCTG	Relation containing connectivity data for the **DKVIG** sensitivity matrix (Character, Input)
DKVIG	Unstructured entity containing the stiffness matrix partitions in a highly compressed format (Character, Input)
[K1GG]	Assembled stiffness matrix in the g-set (Output)
GMMCTG	Relation containing connectivity data for the **DMVIG** sensitivity matrix (Character, Input)
DMVIG	Unstructured entity containing the mass matrix partitions in a highly compressed format (Character, Input)
[M1GG]	Assembled mass matrix in the g-set (Output)

Application Calling Sequence:

None

Method:

The structure of this module resembles that of **EMA1** and is also executed in two passes. In the first pass, **GMKCTG** information is read into core one record at a time. The algorithm is structured to maximize the number of columns of the global stiffness matrix that are assembled at one time. Spill logic is in place if all the columns cannot be assembled at once. For the assembly, subroutine **RQCOR2** performs bookkeeping tasks to expedite the assembly and to determine whether spill will be necessary. Subroutine **ASSEM2** retrieves the **DKVIG** information, performs the assembly in core using the current values of the design variables, and stores the data into **KGG**. When the **GMKCTG** data have been exhausted a check is performed as to whether mass assembly is required. Flags were written by **NLEMA1** on the **INFO** array of the **DKVIG** entity to indicate whether mass assembly is required. If no mass information is required, control is returned to the executive; if it is, the second pass through the module takes place. For the second pass, **DMVIG** and **GMMCTG** data are used to generate the **MGG** matrix. The structure of the assembly operation is otherwise much the same and the **MGG** matrix is computed and stored.

Design Requirements:

1. For **OPTIMIZE** boundary conditions, **EMA2** precedes the optimization boundary condition loop. For **ANALYZE** boundary conditions, the module immediately precedes the loop on analyze boundary conditions and the **NITER** argument is not required. In both cases, **EMA2** must always follow **NLEMA1**.

2. **NITER** must be nonzero for optimization boundary conditions.

Error Conditions:

None

Engineering Application Module: **EMG**

Entry Point: **EMG**

Purpose:

To compute the linear design variable part of element stiffness, mass, thermal load and stress component sensitivities for all structural elements.

MAPOL Calling Sequence:

```
CALL EMG ( NDV, GSIZEB, K6ROT, GLBDES, LOCLVAR, [PTRANS], DESLINK, [SMAT],
           SMATCOL, DVCT, DVSIZE, KELM, MELM,TELM, TREF );
```

NDV	The number of design variables (Integer, Input)
GSIZEB	The size of the structural set (Integer, Input)
K6ROT	Stiffness value for plate element "drilling" degrees of freedom (Real,Input)
GLBDES	Relation of global design variables (Character, Input)
LOCLVAR	Relation containing the relationship between local variables and global variables in the design problem (Input)
[PTRANS]	The design variable linking matrix (Input)
DESLINK	Relation of design variable connectivity from **MAKEST** module containing one record for each global design variable connected to each local variable. (Character, Input)
[SMAT]	Matrix entity containing the linear portion of sensitivity of the stress and strain components to the global displacements (Character, Output)
SMATCOL	Relation containing matrix **[SMAT]** column information (Character,Output)
DVCT	Relation containing the data required for the assembly of the linear design sensitivity matrices (Character, Output)
DVSIZE	Unstructured entity containing memory allocation information on the **DVCT** relation (Character, Output)
KELM	Unstructured entity containing the linear design element stiffness matrix partitions (Character, Output)
MELM	Unstructured entity containing the linear design element mass matrix partitions (Character, Output)
TELM	Unstructured entity containing the linear design element thermal load partitions (Character, Output)
TREF	Unstructured entity containing the linear design element reference temperatures for thermal loads (Character, Output)

Application Calling Sequence:

None

Method:

The **EMG** module performs the linear design variable part of the second phase of the structural element preface operations with the **MAKEST** module performing the first phase. The **NLEMG** module performs the nonlinear design variable part of the second phase. As a result, modules **EMG** and **MAKEST** are very closely related. The first action of the **EMG** module is to determine if design variables and/or thermal loads are defined in the bulk data. If they are, the special actions for design variable linking and thermal stress corrections are taken in the element dependent routines. The **PREMAT** utility to set up the material property data also returns the **SCON** logical flag to denote that there are stress constraints defined in the bulk data. The initialization of the module continues with the retrieval of the **MFORM** data to select lumped or coupled mass matrices in the elements that support both forms. The default is lumped although any **MFORM/COUPLED** (even if **MFORM/LUMPED** also exists) will cause the coupled form to be used. If thermal loads exist, the module prepares the **TREF** entity to be written by the element dependent routines. The **GLBDES** relation is opened and the design variable identification numbers are read into memory. Finally, the **DVCT** entity is opened and flushed and memory is retrieved to be used in the **DVCTLD** submodule to load the **DVCT** relation. The module then calls each element dependent routine in turn. The order in which these submodules are called is very important in that it provides an implicit order for the **MAKEST**, **EMG**, **SCEVAL**, **EDR** and **OFP** modules. That order is alphabetical by connectivity bulk data entry and results in the following sequence:

(1) Bar elements

(2) Scalar spring elements

(3) Linear isoparametric hexahedral elements

(4) Quadratic isoparametric hexahedral elements

(5) Cubic isoparametric hexahedral elements

(6) Scalar mass elements

(7) General concentrated mass elements

(8) Rigid body form of the concentrated mass elements

(9) Isoparametric quadrilateral membrane elements

(10) Quadrilateral bending plate elements

(11) Rod elements

(12) Shear panels

(13) Triangular bending plate elements

(14) Triangular membrane elements

Within each element dependent routine, the **xxxEST** relation for the element is opened and read one tuple at a time. If the **EST** relation indicates that the element is designed, the **DESLINK** data is used to write one set of tuples to the **DVCT** relation for each unique design variable linked to the element. The set of tuples consists of one row for each node to which the element is connected. If the element is not designed, a single set of tuples is written connected to the "zeroth" (implicit) design variable. The element dependent geometry processor is then called to generate the **KELM**, **MELM** and **TELM** entries for the element. Note the **KELM** and **TELM** entities related to nonlinear design stiffnesses are empty, and MELM entries related to nonlinear design mass are empty. These data must be generated before the next call to **DVCTLD** since the **DVCT** forms the directory to all three of these entities. Once all the elements are processed within the current element dependent routine, the **TREF** entity is appended with the vector

of reference temperatures for the current set of elements. Again, the order of these reference temperatures are determined by the sequence listed above and is assumed to hold in other modules. When all the element dependent drivers have been called by the **EMG** module driver, clean up operations begin. The entities that have been open for writing by the element routines are closed, the remaining in-core **DVCT** tuples are written to the data base and the **DVCT** relation is sorted. If there are design variables, the **DVCT** is sorted on the **DVID** attribute and, within each unique **DVID**, by **KSIL**. If there are no design variables (all **DVID**'s are zero), the **DVCT** is sorted only on **KSIL**. Finally, if stress or strain constraints were defined in the bulk data stream, the **SMAT** matrix of constraint sensitivities to the displacements is closed. **SMAT** was opened by the **PREMAT** module when the **SCON** constraint flag was set.

Design Requirements:

1. The **MAKEST** module must have been called prior to the **EMG** module.

Error Conditions:

1. Illegal element geometries and nonexistent material properties are flagged.

Engineering Application Module: **FCEVAL**

Entry Point: **FCEVAL**

Purpose:

To evaluate the current value of all frequency constraints.

MAPOL Calling Sequence:

```
CALL FCEVAL ( NITER, BCID, LAMBDA, CONST );
```

NITER	Design iteration number (Integer, Input)
BCID	User defined boundary condition identification number (Integer, Input)
LAMBDA	Relational entity containing the output from the real eigenanalysis (Input)
CONST	Relation of design constraint values (Character, Output)

Application Calling Sequence:

None

Method:

The **FCEVAL** module first determines if any frequency constraints are applied to the modal analysis in the current boundary condition. If any constraints are applied to the modal analysis, the module proceeds to open the **DCONFRQ** relation to obtain the applied constraints and the **LAMBDA** relation to obtain the computed frequencies. The final initialization task is to open the **CONST** relation to store the computed frequency constraints. The actual computation involves looping through the **DCONFRQ** relation for the current frequency constraint set and conditioning the **LAMBDA** relation to retrieve the results for the modes that are constrained. Having retrieved the mode number and the computed modal frequency from **LAMBDA**, the applied upper or lower bound constraint is computed and stored on the **CONST** relation. Finally, the frequency responses which are required by any user function constraints are also computed.

Design Requirements:

1. The **FCEVAL** module assumes that the current boundary condition is an optimization boundary condition.

Error Conditions:

1. The frequency constraint set referenced by Solution Control does not exist in the **DCONFRQ** relation.
2. The frequency or eigenvector for the constrained mode was not extracted in the real eigenanalysis.
3. The constrained mode is a rigid body mode (zero frequency) and therefore cannot be constrained.

Engineering Application Module: **FLUTDMA**

Entry Point: **FLTDMA**

Purpose:

Assembles the dynamic matrices for the flutter disciplines.

MAPOL Calling Sequence:

```
CALL FLUTDMA ( NITER, BCID, SUB, ESIZE(BC), PSIZE(BC), BGPDT(BC), USET(BC),
               [MAA], [KAA], [TMN(BC)], [GSUBO(BC)], NGDR, LAMBDA, [PHIA],
               [MHHFL(BC,SUB)], [BHHFL(BC,SUB)], [KHHFL(BC,SUB)] );
```

NITER	Design iteration number (Integer, Input)
BCID	User defined boundary condition identification number (Integer, Input)
SUB	Flutter subcase number (Integer, Input)
ESIZE(BC)	Number of extra points for the current boundary condition (Integer, Input), where **BC** represents the MAPOL boundary condition loop index number
PSIZE(BC)	Number of physical degrees of freedom in the current boundary conditions (**GSIZE+ESIZE**) (Integer, Input), where **BC** represents the MAPOL boundary condition loop index number
BGPDT(BC)	Current boundary condition's relation of basic grid point data (expanded to include extra points and any **GDR** scalar points) (Input), where **BC** represents the MAPOL boundary condition loop index number
USET(BC)	Current boundary condition's unstructured entity of set definition masks (expanded to include extra points and any **GDR** scalar points) (Input), where **BC** represents the MAPOL boundary condition loop index number
[MAA]	Mass matrix in the analysis set (Input)
[KAA]	Stiffness matrix in the analysis set (Input)
[TMN(BC)]	Multipoint constraint transformation matrix for the current boundary condition (Input), where **BC** represents the MAPOL boundary condition loop index number
[GSUBO(BC)]	Static condensation or **GDR** reduction matrix for the current boundary condition (Input), where **BC** represents the MAPOL boundary condition loop index number
NGDR	Denotes dynamic reduction in the boundary condition (Input, Integer) = **0** No **GDR** = **-1** **GDR** is used
LAMBDA	Relation of normal mode eigenvalues output from the **REIG** module (Input)
[PHIA]	Matrix of normal mode eigenvectors in the analysis set output from **REIG** (Input)

[MHHFL(BC,SUB)]	Generalized mass matrix for the current flutter subcase in the h-set (normal modes+extra points) including any transfer functions and **M2PP** input (Output), where **BC** represents the MAPOL boundary condition loop index number
[BHHFL(BC,SUB)]	Generalized damping matrix for the current flutter subcase in the h-set (normal modes+extra points) including any transfer functions, **B2PP** input and **VSDAMP** input (Output), where **BC** represents the MAPOL boundary condition loop index number
[KHHFL(BC,SUB)]	Generalized stiffness matrix for the current flutter subcase in the h-set (normal modes+extra points) including any transfer functions **K2PP** input and **VSDAMP** input (Output), where **BC** represents the MAPOL boundary condition loop index number

Application Calling Sequence:

None

Method:

CASE is checked to see if any **FLUTTER** subcases exist for the current boundary condition. If not, control is returned to the MAPOL sequence. If **FLUTTER** subcases exist, the dynamic matrix descriptions for the current subcase (as indicated by the **SUB** input) are brought into memory from **CASE**. Then the **BGPDT** data are read into memory and the **DMAPVC** submodule is called to generate partitioning matrices to expand the input matrices to the p-set from the g-set and to strip off the **GDR** extra points where appropriate. If extra points are defined, the **MAA**, **KAA**, **PHIA**, **TMN** and **GSUBO** are then expanded to include the d-set extra point DOF.

Following the expansion of the input matrices, the direct matrix input **M2PP**, **B2PP** and **K2PP** are assembled and reduced to the direct d-set DOF in the submodule **DMAX2**. Modal transformations occur later in the module. Following the **K2PP** formation, the **VSDAMP** data are set depending on the **DAMPING** selection for the **FLUTTER** subcase. Finally, the **LAMBDA** relation is read into memory to have the modal frequencies available for modal damping computations.

Following all these preparations, the utility submodules **DMAMHH**, **DMABHH** and **DMAKHH** are used to assemble the modal mass, damping and stiffness matrices accounting for all the dynamic matrix options. Control is then returned to the MAPOL program.

Design Requirements:

1. The **FLUTDMA** module is intended to be called once for each **FLUTTER** subcase in the boundary condition. The ordering of subcases is that in the **CASE** relation. Each set of dynamic matrices in the standard sequence is saved in a doubly subscripted set of matrices to be used in sensitivity analysis. It is not necessary to save these matrices unless the sensitivity phase will be performed.

Error Conditions:

1. Missing damping sets called for on the **FLUTTER** entry are flagged.
2. Errors on **TABDMP** entries are flagged.

Engineering Application Module: **FLUTDRV**

Entry Point: **FLUTDR**

Purpose:

MAPOL director for flutter analyses.

MAPOL Calling Sequence:

```
CALL FLUTDRV ( BCID, SUB, LOOP );
```

BCID	User defined boundary condition identification number (Integer, Input)
SUB	Flutter subcase number (ranging from 1 to the total number of **FLUTTER** subcases) of the subcase to be processed in this pass (Integer, Input)
LOOP	Logical flag indicating that more flutter subcases exist in the boundary condition (Logical, Output)

Application Calling Sequence:

None

Method:

The **SUB**'th **FLUTTER** subcase's **TITLE**, **SUBTITLE** and **LABEL** are retrieved from the **CASE** relation and set in the **/OUTPT2/** common for downstream page labeling. If more than **SUB FLUTTER** subcases exist, the **LOOP** flag is set to **TRUE** to tell the MAPOL sequence that more passes through the flutter analysis modules are needed.

Design Requirements:

1. This module is the driver for a set of MAPOL modules that together perform the **FLUTTER** analysis for a subcase. These modules are **FLUTDMA**, **FLUTQHHL** and **FLUTTRAN**.

Error Conditions:

None

Engineering Application Module: **FLUTQHHL**

Entry Point: **FLTQHH**

Purpose:

Processes matrix **QKKL** with normal modes for flutter.

MAPOL Calling Sequence:

```
CALL FLUTQHHL ( NITER, BCID, SUB, ESIZE(BC), PSIZE(BC), [QKKL], [UGTKA],
                [PHIA], USET(BC), [TMN(BC)], [GSUBO(BC)], NGDR, AECOMPU,
                GEOMUA, [PHIKH], [QHHLFL(BC,SUB)], OAGRDDSP );
```

NITER	Design iteration number (Integer, Input)
BCID	User defined boundary condition identification number (Integer, Input)
SUB	Flutter subcase number (Integer, Input)
ESIZE(BC)	Number of extra points for the current boundary condition (Integer, Input), where **BC** represents the MAPOL boundary condition loop index number
PSIZE(BC)	Number of physical degrees of freedom in the current boundary conditions (**GSIZE+ESIZE**) (Integer, Input), where **BC** represents the MAPOL boundary condition loop index number
[QKKL]	Matrix containing a list of k x k complex unsteady aerodynamic matrices for each m-k pair defined by **MKAERO1** and **MKAERO2** entries. These matrices were output from the **AMP** module (Input)
[UGTKA]	The matrix of splining coefficients relating the aerodynamic pressures to forces at the structural grids and relating the structural displacements to the streamwise slopes of the aerodynamic boxes reduced to the a-set DOF (Input)
[PHIA]	Matrix of normal modes eigenvectors in the a-set (Input)
USET(BC)	Current boundary condition's unstructured entity of set definition masks (expanded to include extra points and any **GDR** scalar points) (Input), where **BC** represents the MAPOL boundary condition loop index number
[TMN(BC)]	Multipoint constraint transformation matrix for the current boundary condition (Input), where **BC** represents the MAPOL boundary condition loop index number
[GSUBO(BC)]	Static condensation or **GDR** reduction matrix for the current boundary condition (Input), where **BC** represents the MAPOL boundary condition loop index number
NGDR	Denotes dynamic reduction in the boundary condition (Input, Integer) = **0** No **GDR** = **-1** **GDR** is used

AECOMPU	A relation describing aerodynamic components for the unsteady aerodynamics model. It is used in splining the aerodynamics to the structural model (Input)
GEOMUA	A relation describing the aerodynamic boxes for the unsteady aerodynamics model. The location of the box centroid, normal and pitch moment axis are given. It is used in splining the aerodynamics to the structure and to map responses back to the aerodynamic boxes (Input)
[PHIKH]	A modal tranformation matrix that relates the box-on-box aerodynamic motions to unit displacements of the generalized structural coordinates (modes) (Output)
[QHHLFL(BC,SUB)]	A matrix containing the list of h x h unsteady aerodynamics matrices for the current flutter subcase related to the generalized (modal) coordinates and including control effectiveness (**CONEFFF**), extra points and **CONTROL** matrix inputs (Output), where **BC** represents the MAPOL boundary condition loop index number
OAGRDDSP	A relation containing the structural eigenvectors (generalized DOF) mapped to the aerodynamic boxes for those **AIRDISP** requests in the Solution Control. These terms are the columns of **PHIKH** put in relational form to satisfy the output requests (Output)

Application Calling Sequence:

None

Method:

The **CASE** relation is read to obtain the **SUB**'th flutter subcase parameters: **CONTROL** and **AIRDPRNT**. Then the **FLUTTER** relation is read for the current subcase and the **KLIST** and **EFFID** entries are recovered.

If there is no **CONTROL** matrix, **PHIA** and **UGTKA** matrices are expanded to include dynamic degrees of freedom using the utility module **QHHEXP**. **GDR** scalar points are handled to ensure that the final matrices are in the d-set. If a **CONTROL** matrix does exist, its conformability is checked. The **DMAPVC** utility submodule is used to create partitioning vectors and matrix reduction matrices to allow reduction of the **CONTROL** matrix to the d-set. The **FLCNTR** submodule is then called to append the reduced **CONTROL** matrix to the expanded **UGTKA** matrix. The **PHIKH** matrix is then computed as the product of the expanded **PHIA** and the expanded and **CONTROL**-modified **UGTKA**:

$$[PHIKH] = [PHID]^T[UGTKD]$$

Then, if control effectiveness correction factors are selected for the subcase, the **PHIKH** matrix terms are adjusted by the input factors. This completes the computation of the **PHIKH** output matrix. The input **AIRDISP** output requests are then processed to load the **OAGRDDSP** relation with the generalized displacements on the unsteady aerodynamic geometry.

Finally, the **QKK** matrices that are associated with the user's input Mach number and **KLIST** for the subcase are reduced to the generalized degrees of freedom using the **PHIKH** matrix.

$$[QHHL] = [(PHIKH)]^T[QKKL][PHIKH]$$

The premultiplication takes place in one **MPYAD** and the postmultiplication is done by looping over each reduced frequency in the set, extracting the k columns of each h x k matrix and performing a separate **MPYAD**. The results are then appended onto the output **QHHL**.

Design Requirements:

None

Error Conditions:

1. **CONTROL** matrix errors in conformability are flagged.
2. **CONEFFF** errors are flagged.

Engineering Application Module: **FLUTSENS**

Entry Point: **FLTSTY**

Purpose:

To compute the sensitivities of active flutter constraints in the current active boundary condition.

MAPOL Calling Sequence:

```
CALL FLUTSENS ( NITER, BCID, SUB, LOOP, GSIZEB, NDV, GLBDES, CONST, GMKCT,
                DKVI, GMMCT, DMVI, CLAMBDA, LAMBDA, [QHHLFL(BC,SUB)],
                [MHHFL(BC,SUB)], [BHHFL(BC,SUB)], [KHHFL(BC,SUB)],
                [PHIG(BC)], [AMAT] );
```

NITER	Design iteration number (Integer, Input)
BCID	User defined boundary condition identification number (Integer, Input)
SUB	Flutter subcase number (ranging from 1 to the total number of **FLUTTER** subcases) of the subcase to be processed in this pass. (Integer, Input)
LOOP	Logical flag indicating that more flutter subcases exist in the boundary condition. (Logical, Input)
GSIZEB	The size of the structural set (Integer, Input)
NDV	The number of global design variables (Integer, Input)
GLBDES	Relation of global design variables (Character, Input)
CONST	Relation of constraint values (Character, Input)
GMKCT	Relation containing connectivity data for the **DKVI** sensitivity matrix (Character, Input)
DKVI	Unstructured entity containing the stiffness design sensitivity matrix in a highly compressed format (Character, Input)
GMMCT	Relation containing connectivity data for the **DMVI** sensitivity matrix (Character, Input)
DMVI	Unstructured entity containing the mass design sensitivity matrix in a highly compressed format (Character, Input)
CLAMBDA	Relation containing results of flutter analyses (Character, Input)
LAMBDA	Relational entity containing the output from the real eigenanalysis (Character, Input)
[QHHLFL(BC,SUB)]	Matrix list of modal unsteady aerodynamic coefficients (Input), where **BC** represents the MAPOL boundary condition loop index number
[MHHFL(BC,SUB)]	Modal mass matrix (Input), where **BC** represents the MAPOL boundary condition loop index number
[BHHFL(BC,SUB)]	Modal flutter damping matrix (Input), where **BC** represents the MAPOL boundary condition loop index number
[KHHFL(BC,SUB)]	Modal flutter stiffness matrix (Input), where **BC** represents the MAPOL boundary condition loop index number

[PHIG(BC)]	Matrix of real eigenvectors in the structural set (Input), where **BC** represents the MAPOL boundary condition loop index number
[AMAT]	Matrix of constraint sensitivities (Output)

Application Calling Sequence:

None

Method:

The **FLUTSENS** module is very similar to the **FLUTTRAN** module except that the **CONST** and **CLAMBDA** relations control the execution of the module rather than the **CASE** relation. The module begins by retrieving all the active flutter constraints from the **CONST** relation that are associated with the current subcase (**SUB**) and determining the **CLAMBDA** tuples that correspond to the active constraints. The next task of the module is to prepare for the actual flutter sensitivity analysis by setting up the **FLFACT** bulk data and the **UNMK** data using the **PREFL** and **PRUNMK** utilities, respectively. The generalized mass and damping matrices are then read into memory and converted to single precision, followed by the natural frequencies associated with the computed eigenvectors. Lastly, the generalized stiffness matrix is read in and converted to single precision and the generalized aerodynamic influence coefficients are opened for retrieval. A final operation creates the scratch flutter eigenvector matrices that will be used in the sensitivity evaluation.

For the **FLUTTER** case, a number of tasks are performed to set up for the current Mach number. These consist of the retrieval of the set of m-k pairs for the current **FLUTTER** entry from the **UNMK** data and the set of normal modes that are to be omitted. If modes are omitted, a partitioning vector is created and used to partition the input **PHIG** matrix to include only the desired normal modes. As a final step before the active constraint loop for the current **FLUTTER** set id, the local memory required by the flutter analysis submodules is retrieved.

The module continues with the loop on the **CLAMBDA** tuples associated with the current **FLUTTER** set identification number. The scalar parameters identifying the flutter root are retrieved from **CLAMBDA** and the set of reduced frequencies associated with the **QHLL** matrices for this flutter case are retrieved from the **UNMK** data. The **FA1PKI** submodule is called with this data to compute the interpolation matrix for the **QHLL** matrix list under the **ORIG** curve fit option. Otherwise, the fitting coefficients are computed on the fly within the **QFDRV** family of routines. Then, the subset of the full **QHLL** matrix associated with this flutter analysis is read into core and converted to single precision.

At this point, the imaginary part of the **QZHH** matrix is divided by the reduced frequency. Finally, the **QFDRV** utility is called to generate the **QRS** interpolated aerodynamic influence coefficients for the current flutter eigenvalue. At the same time the **QFDRV** module computes the sensitivity of this matrix to the reduced frequency (**DQRS**). Finally, the flutter eigenmatrix is computed using the **FSUBS** submodule. The corresponding right-hand eigenvector is then computed, the eigenmatrix is transposed and the left-hand vector computed. At this point, the scalar (complex) sensitivities of the mass, damping, stiffness and aerodynamics are computed as outlined in Section 10.3 of the Theoretical Manual. The **FLUTSENS** module performs all these computations using real arithmetic. Finally, the 2 X 2 left-hand side matrices of equation 10-27 of the Theoretical Manual:

$$\begin{bmatrix} DF_{11} & DF_{12} \\ DF_{21} & DF_{22} \end{bmatrix} \begin{Bmatrix} DR \\ DI \end{Bmatrix} = \begin{Bmatrix} P2R * MR - P2I * MI + KR + damping \\ P2R * MI - P2I * MR + KI + damping \end{Bmatrix}$$

are stored and the left- and right-hand eigenvectors packed into a scratch entity. The value ***damping*** is:

$$\begin{Bmatrix} -g * KI \\ g * KR \end{Bmatrix}$$

if structural damping, g, is included.

Simliarly, it is:

$$\begin{Bmatrix} P1R * \frac{g}{\omega_3} * KR - P1I * \frac{g}{\omega_3} * KI \\ P1R * \frac{g}{\omega_3} * KI - P1I * \frac{g}{\omega_3} * KR \end{Bmatrix}$$

if equivalent viscous damping is used at frequency ω_3.

The module then continues with the next active constraint for the current **FLUTTER** entry. Once all the active constraints are treated for the current **FLUTTER** entry, the matrix of left- and right-hand eigenvectors are expanded to physical coordinates using the (partitioned) normal modes matrix. The **FLCSTY** module is then called to complete the solution of the constraint sensitivities to the global design variables. These computations involve the eigenvectors and the mass, damping and stiffness sensitivities to compute the right-hand side of the equations shown in the Theoretical Manual. The flutter response sensitivities which are required by the active user function constraints are also computed in this module. Once the **FLCSTY** module is complete, the **FLUTSENS** module proceeds with the next **FLUTTER** entry with active flutter constraints. When all have been completed, control is returned to the executive.

Design Requirements:

1. The module assumes that at least one active flutter constraint exists in the current boundary condition.

Error Conditions:

None

Engineering Application Module: **FLUTTRAN**

Entry Point: **FLUTAN**

Purpose:

To perform flutter analyses in the current boundary condition and to evaluate any flutter constraints if it is an optimization boundary condition with applied flutter constraints.

MAPOL Calling Sequence:

```
CALL FLUTTRAN ( NITER, BCID, SUB, [QHHLFL(BC,SUB)], LAMBDA, HSIZE(BC),
                ESIZE(BC), [MHHFL(BC,SUB)], [BHHFL(BC,SUB)], [KHHFL(BC,SUB)],
                CLAMBDA, CONST );
```

NITER	Design iteration number (Integer, Input)
BCID	User defined boundary condition identification number (Integer, Input)
SUB	Flutter subcase number (ranging from 1 to the total number of **FLUTTER** subcases) of the subcase to be processed in this pass. (Integer, Input)
[QHHLFL(BC,SUB)]	Matrix list of modal unsteady aerodynamic coefficients (Input)
LAMBDA	Relational entity containing the output from the real eigenanalysis (Input)
HSIZE(BC)	Number of modal dynamic degrees of freedom in the current boundary condition (Input), where **BC** represents the MAPOL boundary condition loop index number
ESIZE(BC)	The number of extra point degrees of freedom in the boundary condition (Integer, Input), where **BC** represents the MAPOL boundary condition loop index number
[MHHFL(BC,SUB)]	Modal mass matrix (Input), where **BC** represents the MAPOL boundary condition loop index number
[BHHFL(BC,SUB)]	Modal flutter damping matrix (Input), where **BC** represents the MAPOL boundary condition loop index number
[KHHFL(BC,SUB)]	Modal flutter stiffness matrix (Input), where **BC** represents the MAPOL boundary condition loop index number
CLAMBDA	Relation containing results of flutter analyses (Character, Output)
CONST	Relation of constraint values (Character, Input)

Application Calling Sequence:

None

Method:

The **FLUTTRAN** module begins by retrieving the flutter discipline entries from the **CASE** relation for the current boundary condition. If the boundary condition is an optimize boundary condition, the **CONST** and **CLAMBDA** relations are opened to store the constraint and root extraction data needed for the optimization task. For analysis boundary conditions, the hidden entities **FLUTMODE** and **FLUTREL** are opened and initialized to prepare for possible flutter mode shape storage. These mode shapes are stored so that the **OFPDISP** module can satisfy flutter mode shape print requests.

The next task of the module is to prepare for the actual flutter analysis by setting up the **FLFACT** bulk data and the **UNMK** data using the **PREFL** and **PRUNMK** utilities, respectively. Then the reference unsteady aerodynamic model data is retrieved from the **AERO** relation. Lastly, the velocity conversion factor, if one has been defined, is read from the **CONVERT** relation. The generalized mass and damping matrices are then read into memory and converted to single precision, followed by the natural frequencies associated with the computed eigenvectors. Lastly, the generalized stiffness matrix is read in and converted to single precision and the generalized aerodynamic influence coefficients are opened for retrieval. This completes the preparations for the flutter discipline loop.

For the **SUB**'th flutter discipline requested in the **CASE** relation, a number of tasks are performed to set up for the Mach number requested on the **FLUTTER** entry. These consist of the retrieval of the set of m-k pairs for the current **FLUTTER** entry from the **UNMK** data and the lists velocities (which are converted to the proper units, if necessary) and densities. If the boundary condition is an optimization boundary condition, the table of required damping values is prepared using the **PRFCON** utility. Lastly, the set of normal modes that are to be omitted are retrieved and the data prepared to perform the "partitioning" of the generalized matrices. As a final step before processing the current **FLUTTER** entry, the local memory required by the flutter analysis submodules is retrieved.

The subset of m-k pairs in the **QHLL** matrix list for the current Mach number is determined and the set of associated reduced frequencies determined. The **FA1PKI** submodule is called with this data to compute the interpolation matrix for the **QHLL** matrix list if the **ORIG** curve fit is used. Otherwise, the fitting coefficients are computed on the fly in the **QFDRV** module. Then, the subset of the full **QHLL** matrix associated with this flutter analysis is read into core and converted to single precision. At this point, the imaginary part of the **QZHH** matrix is divided by the reduced frequency. Finally, the Mach number dependent memory blocks are retrieved and the inner-most analysis loop on the density ratios is begun.

For each density ratio associated with the Mach number for the current flutter analysis discipline, the **FLUTTRAN** module performs the flutter analysis. There are two distinct paths through the inner loop: one for optimization and one for analysis. They differ in that the analysis loop refines the set of user selected velocities to find a flutter crossing, while the optimization path computes the flutter eigenvalues only at the user specified velocities and computes the corresponding flutter constraint value based on the required damping table. Once all the loops have been completed, the module computes the flutter responses which are required by any user function constraints, and then returns control to the executive.

Design Requirements:

1. The module assumes that at least one flutter subcase exists in the current boundary condition.

Error Conditions:

1. Referenced data on **FLUTTER** entries that do not exist on the data base are flagged and the execution is terminated.

Engineering Application Module: **FNEVAL**

Entry Point: **FNEVAL**

Purpose:

Evaluates the current values of user functional constraints.

MAPOL Calling Sequence:

```
CALL FNEVAL ( NITER, CONST );
```

NITER	Design iteration number (Integer,Input)
CONST	Relation of constraint values (Character,Output)

Application Calling Sequence:

None

Method:

This module first computes the user defined objective function value if it is required. This objective function value is stored in relation **CONST** with the **OBJECTIVE** indication flag set. The function value is set to be "active" so that its required response sensitivities will be computed thereafter. Then, all instances invoked by the objective are computed. They are treated as subcase independent user functional constraints.

Finally, case dependent user functional constraints are computed for each subcase for which functional constraints have been specified. All user function values are computed using the evaluation utilities. These utilities retrieve all required response function values.

Design Requirements:

1. All response functions must be computed prior to this module.

Error Conditions:

None

Engineering Application Module: **FPKEVL**

Entry Point: **FPKEVL**

Purpose:

Compiles the FUNCTION packet and instantiates the user functions that have been invoked by Solution Control.

MAPOL Calling Sequence:

```
CALL FPKEVL ( EIDTYPE );
```

EIDTYPE	Relation containing element identification numbers and corresponding element type (Character,Input)

Application Calling Sequence:

None

Method:

This module compiles the function packet statements and loads the compiled information onto the CADDB database. Compilation determines the validity of each function in terms of its syntax and the other functions it may use. Once the validity of the functions is determined, each function is instantated. Instantiation determines that the supporting Bulk Data, if any, is present on the database and the actual number of instances. The instantiation process creates the data structures that describe each constraint. These data structures are then used by ASTROS to request the computation of the constituent responses.

Design Requirements:

1. A Function packet must be included in the input data stream.

Error Conditions:

1. Syntax errors and inconsistent or illegal function requests are flagged and the execution is terminated.

Engineering Application Module: **FREDUCE**

Entry Point: **FREDUC**

Purpose:

To reduce the symmetric or asymmetric f-set stiffness, mass and/or loads matrix to the a-set if there are omitted degrees of freedom.

MAPOL Calling Sequence:

```
CALL FREDUCE ( [KFF], [PF], [PFOA(BC)], SYM, [KOOINV(BC)], [KOOU(BC)],
               [KAO(BC)], [GSUBO(BC)], [KAA], [PA], [PO], USET(BC) );
```

[KFF]	Optional Stiffness or mass matrix to be reduced (Input)
[PF]	Optional loads matrix to be reduced (Input)
[PFOA(BC)]	The partitioning vector splitting the free degrees of freedom into the analysis set and the omitted degrees of freedom (Input), where **BC** represents the MAPOL boundary condition loop index number
SYM	Optional symmetry flag; **=1** if **KFF** is not symmetric (Integer, Input)
[KOOINV(BC)]	Matrix containing the inverse of **KOO** for symmetric stiffness matrices or the lower triangular factor of **KOO** for asymmetric matrices (Output), where **BC** represents the MAPOL boundary condition loop index number
[KOOU(BC)]	Optional matrix containing the upper triangular factor of **KOO** for asymmetric stiffness matrices (Output), where **BC** represents the MAPOL boundary condition loop index number
[KAO(BC)]	Optional matrix containing the off-diagonal partition of **KFF** required for recovery when **KFF** is asymmetric (Output), where **BC** represents the MAPOL boundary condition loop index number
[GSUBO(BC)]	Matrix containing the static condensation transformation matrix (Input and Output), where **BC** represents the MAPOL boundary condition loop index number
[KAA]	The stiffness matrix in the analysis set degrees of freedom (Output)
[PA]	The loads matrix in the analysis set degrees of freedom (Output)
[PO]	Matrix containing the loads on the omitted degrees of freedom (Output)
USET(BC)	The unstructured entity defining structural sets (Character, Input), where **BC** represents the MAPOL boundary condition loop index number

Application Calling Sequence:

None

Method:

FREDUCE module begins by checking if the **KFF** argument is nonblank. If so, the reduction by static condensation is performed in one of two ways depending on the **SYM** flag. If the **SYM** flag is zero or omitted from the calling sequence the following operations are performed:

$$[\mathbf{KFF}] \rightarrow \begin{bmatrix} \mathbf{KOO} & \mathbf{KOA} \\ \mathbf{KOA}^{T} & \overline{\mathbf{KAA}} \end{bmatrix}$$

[KOOINV] = **[KOO]**$^{-1}$ symmetric decomposition

[GSUBO] = -**[KOOINV]** **[KOA]** symmetric Forward-Backward Substition

$$\begin{bmatrix} \mathbf{KAA} \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{KAA}} \end{bmatrix} + \begin{bmatrix} \mathbf{KOA} \end{bmatrix}^{T} \begin{bmatrix} \mathbf{GSUBO} \end{bmatrix}$$

The **KOOINV**, **GSUBO** and **KAA** arguments must be nonblank in the calling sequence. If the **SYM** flag is nonzero in the calling sequence the following operations are performed:

$$\begin{bmatrix} \mathbf{KFF} \end{bmatrix} \rightarrow \begin{bmatrix} \mathbf{KOO} & \mathbf{KOA} \\ \mathbf{KAO} & \overline{\mathbf{KAA}} \end{bmatrix}$$

[KOOINV] and **[KOOU]** are the Lower and Upper triangular factors of **[KOO]**

[GSUBO] = - **[KOO]**$^{-1}$**[KOA]** asymmetric Forward-Backward Substition

[KAA] = **[$\overline{\text{KAA}}$]** + **[KAO]** **[GSUBO]**

The **KOOINV**, **KOOU**, **KAO**, **GSUBO** and **KAA** arguments must be nonblank in the calling sequence. Note that **KAO** is required since the asymmetric nature of **KFF** prohibits the transpose operation used in the symmetric case. The module then checks if **PF** is nonblank. If so, the loads matrix reduction is performed. Once again, there are two paths depending on the symmetry flag. If **SYM** is zero (symmetric), the following operations are performed:

$$\begin{bmatrix} \mathbf{PF} \end{bmatrix} \rightarrow \begin{Bmatrix} \mathbf{PO} \\ \overline{\mathbf{PA}} \end{Bmatrix}$$

[SCR1] = **[PO]**^{T}**[GSUBO]**

[SCR2] = **[SCR1]**^{T}

[PA] = **[$\overline{\text{PA}}$]** + **[SCR2]**

With the odd order of operations dictated by efficiency considerations in the matrix operations. Note that the **GSUBO**, **PA** and **PO** arguments must be nonblank with the **GSUBO** argument an input if the stiffness matrix was not simultaneously reduced. If **SYM** is nonzero (asymmetric), the following operations are performed:

$$\left[\mathbf{PF} \right] \rightarrow \left\{ \begin{matrix} \mathbf{PO} \\ \overline{\mathbf{PA}} \end{matrix} \right\}$$

$$\mathbf{[SCR1]} = \mathbf{[KOO]}^{-1}\ \mathbf{[PO]} \quad \text{(Asymmetric FBS)}$$

$$\mathbf{[PA]} = \mathbf{[\overline{PA}]} + \mathbf{[KAO]}\,\mathbf{[SCR1]}$$

Note that the **KOOINV**, **KOOU**, **KAO**, **PA** and **PO** arguments must be supplied with the **KOOINV**, **KOOU**, **and KAO** arguments input if the (asymmetric) stiffness matrix is not being reduced in the same call.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **FREQSENS**

Entry Point: **FQCSTY**

Purpose:

To compute the sensitivities of active frequency constraints in the current active boundary condition.

MAPOL Calling Sequence:

```
CALL FREQSENS ( NITER, BCID, NDV, GLBDES, CONST, LAMBDA, GMKCT, DKVI, GMMCT,
                DMVI, [PHIG(BC)], [AMAT] );
```

NITER	Design iteration number (Integer, Input)
BCID	User defined boundary condition identification number (Integer, Input)
NDV	Number of design variables (Integer, Input)
GLBDES	Relation of global design variables (Character, Input)
CONST	Relation of constraint values (Character, Input)
LAMBDA	Relational entity containing the output from the real eigenanalysis (Input)
GMKCT	Relation containing connectivity data for the **DKVI** sensitivity matrix (Character, Input)
DKVI	Unstructured entity containing the stiffness design sensitivity matrix in a highly compressed format (Character, Input)
GMMCT	Relation containing connectivity data for the **DMVI** sensitivity matrix (Character, Input)
DMVI	Unstructured entity containing the mass design sensitivity matrix in a highly compressed format (Character, Input)
[PHIG(BC)]	Matrix of eigenvectors for the current boundary condition (Input), where **BC** represents the MAPOL boundary condition loop index number
[AMAT]	Matrix containing the sensitivities of the constraints to the design variables (Output)

Application Calling Sequence:

None

Method:

This module first computes the frequency response sensitivities which are required by the active user function constraints. Then it obtains design variable information from **GLBDES**, frequency constraint information from **CONST** and eigenvalue information from **LAMBDA**. Space is reserved for design sensitivity matrices and then the number of eigenvectors that can be held in core simultaneously is determined. Spill logic is provided if this number is less than the number of eigenvectors that have eigenvalue constraints.The eigenvectors are read into core and a loop on the design variables brings the connectivity data into core. Calls to **TUNMLS/D** perform the required triple matrix products involving the eigenvector and the design sensitivity matrices. This information is required to form the frequency constraint information, which is written to the **AMAT** matrix.

Design Requirements:

1. The module is only called if there are active frequency constraints and therefore must follow the **ABOUND** module.

2. The **DESIGN** module makes the assumption that data were written to **AMAT** from this module prior to any subcase dependent sensitivities.

Error Conditions:

None

Engineering Application Module: **FSD**

Entry Point: **FSDDRV**

Purpose:

To perform redesign by Fully Stressed Design (**FSD**) methods based on the set of applied stress constraints. All other applied constraints are ignored.

MAPOL Calling Sequence:

```
CALL FSD ( NDV, NITER, FSDS, FSDE, MPS, OCS, ALPHA, CNVRGLIM, GLBDES,
           LOCLVAR, [PTRANS], CONST, APPCNVRG, CTL, CTLMIN, DESHIST );
```

NDV	The number of global design variables (Integer, Input)
NITER	Iteration number for the current design iteration (Integer, Input)
FSDS	The first iteration to use **FSD** (Integer, Input)
FSDE	The last iteration to use **FSD** (Integer, Input)
MPS	The first iteration to use math programming (Integer, Input)
OCS	The first iteration to use optimality criteria (Integer, Input)
ALPHA	Exponential move limit for the **FSD** algorithm (Real, Input)
CNVRGLIM	Relative percent change in the objective function that indicates approximate problem convergence (Real, Input)
GLBDES	Relation of global design variables (Character, Input)
LOCLVAR	Relation containing the relationship between local variables and global variables in the design problem (Character, Input)
[PTRANS]	The design variable linking matrix (Character, Input)
CONST	Relation of constraint values (Character, Input)
APPCNVRG	The approximate problem converge flag (Logical, Output) = **FALSE** if not converged = **TRUE** if converged in objective function value and design vector move
CTL	Tolerance for indicating an active constraint (Real, Output)
CTLMIN	Tolerance for indicating a violated constraint (Real, Output)
DESHIST	Relation of design iteration information (Character, Output)

Application Calling Sequence:

None

Method:

The first task performed in the **FSD** module is to determine if the **FSD** option is to be used. The assumption of the module is that the Solution Control **STRATEGY** requests have been satisfied by the MAPOL sequence such that, if **FSD** is called, **FSD** has been requested by the user for this iteration.

The module checks that the **ALPHA** parameter is a legal value (**>0.0**). If it is not, the default value of 0.50 is used. Then **FSD** brings the required data into memory. These data consist of the local design

variable data (in the **PTRANS**, **LOCLVAR** and **GLBDES** entities), which are accessed through the design variable utility module **PRELDV** with entry points **LDVPNT** and **GETLDV**. Finally, the **CONST** relation tuples associated with the stress constraints are retrieved. If no stress constraints are found, the module cannot do any resizing and so modifies the MAPOL control parameters **FSDS**, **FSDE**, **MPS** and **OCS** as outlined below to prevent the further use of **FSD** in subsequent iterations.

If the appropriate constraints were found, the module loops through each local design variable and determines which (if any) stress constraint is associated with that variable. When the matching constraint is found, the new local variable is computed from:

$$t_{new} = (g + 1.0)^{\alpha}$$

If any shape function linked local variables are encountered during this phase, the starting and ending iterations (**FSDS** and **FSDE**) and the appropriate other starting iteration number (the lesser of **MPS** and **OCS**) are modified such that **FSD** will not be called again. Then execution is returned to the executive. This prevents any further attempts to use **FSD** with the shape function linking and directs that the current iteration be performed used the appropriate alternative method. A warning is given and the execution continues.

Once the vector of new local variables are retrieved, the **PTRANS** data is brought into memory along with the **GLBDES** data. The **GLBDES** data are used to reset any local variable values that are outside their valid ranges to maximum or minimum gauge. Then the new vector of global variables are computed as:

$$v_{new} = \max_{P_i} \left(\frac{t_{new}}{P_i} \right)$$

These constitute the new design from the **FSD** algorithm and are stored back to the **GLBDES** relation. The **DESHIST** relation is updated and an informational message indicating the changes in the objective function is written. The active and violated constraint tolerances are set to their **FSD** default values: **CTL=-1.0 x** 10^{-3} and **CTLMIN=5.0x**10^{-3}. This completes the action of the **FSD** module.

Design Requirements:

1. Only stress constraints (strain constraints are excluded) are considered in the **FSD** module. If none are found, the module terminates cleanly with the **FSD** selection flags reset to avoid any further **FSD** cycles.

2. Shape function design variable linking causes the module to terminate cleanly with the **FSD** selection flags reset to avoid any further **FSD** cycles.

Error Conditions:

None

Engineering Application Module: **GDR1**

Entry Point: **GDRDR1**

Purpose:

To compute the shifted stiffness matrix and the rigid body transformation matrix **[GGO]** to be used in phase 2 of Generalized Dynamic Reduction.

MAPOL Calling Sequence:

```
CALL GDR1 ( [KOO], [MOO], [KSOO], [GGO], LKSET, LJSET, NEIV, FMAX, BCID,
            BGPDT(BC), USET(BC), NOMIT, LSIZE );
```

[KOO]	Stiffness matrix in the o-set (Input)
[MOO]	Mass matrix in the o-set (Input)
[KSOO]	Shifted **KOO** matrix (Output)
[GGO]	Matrix to compute displacements at the g-set due to displacements at the origin (Output)
LKSET	Length of the k-set vectors, **LKSET = -1** if there is no k-set (Integer, Output)
LJSET	Length of the j-set, **LJSET= -1** if there is no jset
NEIV	Computed number of eigenvalues below **FMAX** (Integer, Output)
FMAX	Maximum frequency of interest. This is user supplied through the **DYNRED** entry, but may be modified after output to give the desired number of eigenvalues on input to **GDR2** (Real,Output)
BCID	User defined boundary condition identification number (Integer, Input)
BGPDT(BC)	Relation of basic grid point coordinate data (Input), where **BC** represents the MAPOL boundary condition loop index number
USET(BC)	The unstructured entity defining structural sets (Input), where **BC** represents the MAPOL boundary condition loop index number
NOMIT	The number of DOF in the o-set (Integer, Input)
LSIZE	The number of DOF in the L-set (Integer, Input)

Application Calling Sequence:

None

Method:

The module begins by calling subroutine **GDR1S** to input bulk data information. **NEIV,** the number of eigenvalues, is then determined using **FMAX** and the Sturm sequence theorem. The **LJSET** parameter is computed as a combination of structural DOF in the a-set plus any user input nonstructural DOF. The **LKSET** parameter is specified to be **1.5 *NEIV** and the shift parameter is computed based on **FMAX**, **LKSET** and the machine precision. The shifted stiffness matrix is then computed, the **GGO** matrix is computed and control is returned to the executive.

Design Requirements:

1. This module is an alternative to Guyan reduction and therefore parallels the reduction to the a-set.

Error Conditions:

1. j-set DOF have been constrained
2. o-set does not exist
3. Only a subset of roots are guaranteed to be accurate.

Engineering Application Module: **GDR2**

Entry Point: **GDRDR2**

Purpose:

To compute the orthogonal basis **[PHIOK]** for the subspace to be used in phase 3 of Generalized Dynamic Reduction.

MAPOL Calling Sequence:

```
CALL GDR2      ( [LSOO], [MOO], [PHIOK], LKSET, LJSET, NEIV, FMAX, BCID );
```

[LSOO]	Decomposed shifted stiffness matrix (Input)
[MOO]	Mass matrix in the o-set (Input)
[PHIOK]	Matrix of approximate vectors (Output)
LKSET	Length of the k-set vectors (Integer, Input)
LJSET	Not used
NEIV	Number of eigenvalues below **FMAX** (Integer, Input)
FMAX	Maximum frequency of the **NEIV** eigenvalues (Real,Input)
BCID	User defined boundary condition identification number (Integer, Input)

Application Calling Sequence:

None

Method:

After performing initialization tasks, random starting vectors are generated and an iteration procedure is performed to obtain an initial set of solution vectors. These solution vectors are transformed into a orthogonal base for the approximate vectors. If an insufficient number (<< **LKSET**) vectors are generated by this process, additional solution vectors are obtained and transformed.

Design Requirements:

1. This module follows **GDR1** and a decomposition of **KSOO** into **LSOO**.
2. If **LKSET** is zero in the standard MAPOL sequence, **GDR2** is not called.

Error Conditions:

None

Engineering Application Module: **GDR3**

Entry Point: **GDRDR3**

Purpose:

To compute the transformation matrix **[GSUBO]** for Generalized Dynamic Reduction.

MAPOL Calling Sequence:

```
CALL GDR3 ( [KOO], [KOA], [MGG], [PHIOK], [TMN(BC)], [GGO], [PGMN(BC)],
            [PNSF(BC)], [PFOA(BC)], [GSUBO(BC)], BGPDT(BC), USET(BC), LKSET,
            LJSET, ASIZE, GNORM, BCID );
```

[KOO]	Stiffness matrix in the o-set (Input)
[KOA]	Partition of the stiffness matrix (Input)
[MGG]	Mass matrix in the g-set (Input)
[PHIOK]	Matrix of approximate eigenvectors (Input)
[TMN(BC)]	Matrix relating m-set and n-set DOF's (Input), where **BC** represents the MAPOL boundary condition loop index number
[GGO]	Rigid body transformation matrix (Input)
[PGMN(BC)]	Partitioning vector from g to m and n-sets (Input), where **BC** represents the MAPOL boundary condition loop index number
[PNSF(BC)]	Partitioning vector from n to s and f-sets (Input), where **BC** represents the MAPOL boundary condition loop index number
[PFOA(BC)]	Partitioning vector from f to o and a-sets (Input), where **BC** represents the MAPOL boundary condition loop index number
[GSUBO(BC)]	General transformation matrix for dynamic reduction (Output), where **BC** represents the MAPOL boundary condition loop index number
BGPDT(BC)	Relation of basic grid point coordinate data (Input), where **BC** represents the MAPOL boundary condition loop index number
USET(BC)	The unstructured entity defining structural sets (Input), where **BC** represents the MAPOL boundary condition loop index number
LKSET	Length of the k-set (Integer, Input)
LJSET	Length of the j-set (Integer, Input and Output)
ASIZE	The number of DOF's in the A-set (Integer, Input)
GNORM	The sum of **LKSET** and **LJSET** (Integer, Output)
BCID	Boundary condition identification number (Integer, Input)
BC	Boundary condition index number (Integer, Input)

Application Calling Sequence:

None

Method:

The module calculates an overall transformation matrix which relates DOF's in the a-, j- and k-sets to the o-set. The task is simplified if some of the sets are empty.

Design Requirements:

1. This module must follow **GDR1**.

2. If **LKSET** is nonzero, **GDR2** must also have been called.

Error Conditions:

None

Engineering Application Module: **GDR4**

Entry Point: **GDRDR4**

Purpose:

To compute updated transformations between displacement sets. Useful for data recovery from Generalized Dynamic Reduction.

MAPOL Calling Sequence:

```
CALL GDR4 ( BCID, GSIZE, PSIZE(BC), LKSET, LJSET, [PGMN(BC)], [TMN(BC)],
            [PNSF(BC)], [PFOA(BC)], [PARL(BC)], [PGDRG(BC)], [PAJK],
            [PFJK], BGPDT(BC), USET(BC) );
```

BCID	User defined boundary condition identification number (Integer, Input)
GSIZE	Length of the g-set vector (Integer, Input)
PSIZE(BC)	The size of the physical set for the current boundary condition. (Integer, Input), where **BC** represents the MAPOL boundary condition loop index number
LKSET	Length of the k-set (Integer, Input)
LJSET	Length of the j-set (Integer, Input)
[PGMN(BC)]	Partitioning vector from g to m and n-sets (Input), where **BC** represents the MAPOL boundary condition loop index number
[TMN(BC)]	Matrix relating m-set and n-set DOF's (Input), where **BC** represents the MAPOL boundary condition loop index number
[PNSF(BC)]	Partitioning vector from n to s and f-sets (Input), where **BC** represents the MAPOL boundary condition loop index number
[PFOA(BC)]	Partitioning vector from f to o and a-sets (Input), where **BC** represents the MAPOL boundary condition loop index number
[PARL(BC)]	Modified partitioning vector to partition the a-set to r and l-sets (Output), where **BC** represents the MAPOL boundary condition loop index number
[PGDRG(BC)]	A partitioning vector that removes the additional **GDR** scalar points from the g-set sized displacement and acceleration vectors. (Output), where **BC** represents the MAPOL boundary condition loop index number
[PAJK]	Partitioning vector to divide the a-set DOFs that may include **GDR** generated scalar points into the original a-set DOF's. (Output)
[PFJK]	Partitioning vector to divide the f-set DOFs that may include **GDR** generated scalar points into the original f-set DOF's. (Output)
BGPDT(BC)	**GDR** modified relation of basic grid point coordinate data which, on output, will include the scalar points generated by **GDR**. (Input and Output), where **BC** represents the MAPOL boundary condition loop index number
USET(BC)	**GDR** modified unstructured entity defining structural sets which, on output, will include the scalar points generated by **GDR**. (Input and Output), where **BC** represents the MAPOL boundary condition loop index number

Application Calling Sequence:

None

Method:

The module computes the partitioning matrix **PGDRG** to allow reduction of the downstream g-set matrices to be only the original g-set (before **GDR** scalar points were added). Similarly the **PFJK** partitioning vector does the same for f-set matrices. The **USET** and **BGPDT** entities are updated to include the **GDR** extra points (which are assigned external id's greater than any existing scalar points and internal id's greater than the g-size). These degrees of freedom will belong to the k- and/or j-sets. Lastly, a modified **PARL** partitioning vector is also computed in which the a-set are the generalized **GDR** degrees of freedom (scalar points and/or physical DOF) and the r-set are the support point DOFs.

Design Requirements:

1. This is the final module in the **GDR** sequence

Error Conditions:

None

Engineering Application Module: **GDVGRAD**

Entry Point: **GDVGRD**

Purpose:

Computes the sensitivity of design variable intrinsic functions to the changes of design variables.

MAPOL Calling Sequence:

```
CALL GDVGRAD ( NITER, NDV, CONST, GLBDES );
```

NITER	Design iteration number (Integer,Input)
NDV	Number of design variables (Integer,Input)
CONST	Relation of constraint values (Character,Input)
GLBDES	Relation of global design variables (Character,Input)

Application Calling Sequence:

None

Method:

This module first determines the active user functional constraints at the current design iteration. Then the design variable intrinsic entity is searched to find the design variable intrinsic functions which are required. The sensitivities to the design variables are computed for those functions and stored into the design variable intrinsic function sensitivity relation and matrix entities.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **GDVRESP**

Entry Point: **GDVRSP**

Purpose:

Computes the design variable intrinsic function values.

MAPOL Calling Sequence:

```
CALL GDVRESP ( NITER, NDV, GLBDES );
```

NITER	Design iteration number (Integer,Input)
NDV	Number of design variables (Integer,Input)
GLBDES	Relation of global design variables (Character,Input)

Application Calling Sequence:

None

Method:

This module searches through all design variable intrinsic entity entries and obtains all design variable identification numbers which are required to evaluate user functions. The values of those required design variables are obtained from relation **GLBDES** and stored into the design variable intrinsic response entity.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **GPSP**

Entry Point: **GPSP**

Purpose:

Processes the n-set stiffness matrix to identify singularities, and, if requested, automatically removes them.

MAPOL Calling Sequence:

```
CALL GPSP ( NITER, BCID, NGDR, [KNN], BGPDT(BC), [YS(BC)],
            USET(BC), GPST(BC) );
```

NITER	Design iteration number (Integer,Input)
BCID	User defined boundary condition identification number (Integer, Input)
NGDR	Denotes dynamic reduction in the boundary condition (Input,Integer) = **0** No GDR is requested = **-1** GDR has been requested
[KNN]	A partition of the **KGG** matrix which contains global stiffness matrix (Character,Input)
BGPDT(BC)	Relation of basic grid point coordinates (Character,Input), where **BC** represents the MAPOL boundary condition loop index number
[YS(BC)]	The vector of enforced displacements (Character,Input), where **BC** represents the MAPOL boundary condition loop index number
USET(BC)	Unstructured entity defining structural sets for each degree of freedom (Character,Input), where **BC** represents the MAPOL boundary condition loop index number
GPST(BC)	Unstructured entity contains the grid point singularity table information (Character,Output), where **BC** represents the MAPOL boundary condition loop index number

Application Calling Sequence:

None

Method:

In this module, for each grid or scalar point the following basic processing is done. For grid points two 3 x 3 stiffness matrices are processed and for scalar points the single diagonal stiffness term is processed. Each of these is checked for potential singularities by performing an eigenvalue analysis. If any singularities are found and the degree of freedom in question is not in the s-set, m-set or connected to an MPC equation, then a Single Point Constraint (SPC) is generated. This module processes the n-set matrix and will assure that any singularities left are removed using the same basic processing occurs except only SPC's are generated and **RG** is purged so that no MPC connection processing occurs.

Design Requirements:

None

Error Conditions:

If a non-positive definite partition of the stiffness matrix is detected during AUTOSPC processing, or if the eigenanalysis fails to converge, the program terminates.

Engineering Application Module: **GPWG**

Entry Point: **GPWG**

Purpose:

Grid point weight generator.

MAPOL Calling Sequence:

```
CALL GPWG ( NITER, BCID, GPWGGRID, [MGG], OGPWG );
```

NITER	Design iteration number (Optional, Integer, Input)
BCID	User defined boundary condition identification number (Integer, Input)
GPWGGRID	Relation containing the data from the **GPWG** Bulk Data entries (Input)
[MGG]	Mass matrix in the g-set (Input)
OGPWG	Relation of Grid Point Weight Generation Output (Output)

Application Calling Sequence:

None

Method:

The existence of the **MGG** matrix is checked first. If it does not exist or has no columns, control is returned to the MAPOL sequence without error. Then the **CASE** relation is read for the current boundary condition to determine if any **GPWG** print or punch requests exist. If not, the module terminates.

The invariant basic grid point data are read from **BGPDT** and checked to ensure that at least one grid point exists. If all the points are scalar points, the module terminates without warning. Then the **CONVERT/MASS** entry is recovered (if one exists) to allow output of the Grid Point Weight from the mass matrix. Then the coordinates of the reference point are found from the first **GPWG** entry in the **GPWGGRID** relation or are set to {**0.0,0.0,0.0**}.

The grid point weight is then computed and the results are stored on the **OGPWG** relation. If a **PRINT** request exists for the current design iteration or analyse boundary condition, the results are read from **OGPWG** and printed to the output file.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **GREDUCE**

Entry Point: **GREDUC**

Purpose:

To reduce the symmetric g-set stiffness, mass or loads matrix to the n-set if there are multipoint constraints in the boundary condition.

MAPOL Calling Sequence:

```
CALL GREDUCE ( [KGG], [PG], [PGMN(BC)], [TMN(BC)], [KNN], [PN] );
```

[KGG]	Optional matrix containing the global stiffness or mass matrix to be reduced (Input)
[PG]	Optional matrix containing the global applied loads to be reduced (Input)
[PGMN(BC)]	The partitioning vector splitting the structural degrees of freedom into the independent and the multipoint constraint degrees of freedom (Input), where **BC** represents the MAPOL boundary condition loop index number
[TMN(BC)]	The transformation matrix for multipoint constraints (Input), where **BC** represents the MAPOL boundary condition loop index number
[KNN]	Optional matrix containing the reduced **KGG** matrix for the independent degrees of freedom (Output)
[PN]	Optional matrix containing the reduced **PG** matrix for the independent degrees of freedom (Output)

Application Calling Sequence:

None

Method:

The **GREDUCE** utility module performs the multipoint constraint reduction on the stiffness and/or mass and/or loads matrix based on the presence or absence of input arguments. The only required arguments are the partitioning vector **PGMN** and the rigid body transformation matrix **TMN**. If the **KGG** argument is not omitted, the following operations are performed using the large matrix utilities:

$$\begin{bmatrix} KGG \end{bmatrix} \rightarrow \begin{bmatrix} KMM & KMN \\ KNM & \overline{KNN} \end{bmatrix}$$

$$[SCR1] = [\overline{KNN}] + [KNM]\,[TMN]$$

$$[SCR2] = [SCR1] + [TMN]^T[KMN]$$

$$[SCR1] = [KMM]\,[TMN]$$

$$[SCR2] = [SCR1] + [TMN]^T[KMN]$$

$$[KNN] = [SCR2] + [TMN]^T[KMN]$$

These operations require the creation of four scratch matrix entities for the intermediate results and the partitions of the **KGG** matrix.

If the **PG** argument is not omitted, the following operations are performed using the large matrix utilities:

$$\left[\ \mathbf{PG}\ \right] \rightarrow \left\{ \begin{array}{c} \mathbf{PM} \\ \overline{\mathbf{PN}} \end{array} \right\}$$

$$[\mathbf{PN}] = [\overline{\mathbf{PN}}] + [\mathbf{TMN}]^{\mathbf{T}}[\mathbf{PM}]$$

These operations require the use of two scratch matrix entities for the partitions of the **PG** matrix. When both **KGG** and **PG** are reduced, the scratch partition matrices are shared.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **GTLOAD**

Entry Point: **GTLOAD**

Purpose:

To assemble the current static applied loads matrix for any statics subcases in the current boundary condition.

MAPOL Calling Sequence:

```
CALL GTLOAD ( NITER, BCID, GSIZE, BGPDT(BC), GLBDES, SMPLOD, [DPTHVI],
              [DPTHVD], [DPGRVI], [DPGRVD], [PG], OGRIDLOD );
```

NITER	Design iteration number (Integer, Input)
BCID	User defined boundary condition identification number (Integer, Input)
GSIZE	The size of the structural set (Integer, Input)
BGPDT(BC)	Relation of basic grid point coordinate data (Input), where **BC** represents the MAPOL boundary condition loop index number
GLBDES	Relation of global design variables (Input)
SMPLOD	Unstructured entity of simple load vector information (Input)
[DPTHVI]	Matrix entity containing the linear thermal load sensitivities (Input)
[DPTHVD]	Matrix entity containing the nonlinearly designed thermal loads (Character, Input)
[DPGRVI]	Matrix entity containing the linear gravity load sensitivities (Input)
[DPGRVD]	Matrix entity containing the nonlinearly designed gravity loads (Character, Input)
[PG]	The matrix of applied loads in the global structural set (Output)
OGRIDLOD	Relation of loads on structural grid points. (Output)

Application Calling Sequence:

None

Method:

The **CASE** relation tuples for the current boundary condition are brought into memory to obtain the mechanical, thermal and/or gravity simple load identification numbers for each **STATICS** discipline. The **LOAD** bulk data relation is also read into memory to process combined simple loads requests. Finally, the **SMPLOD** data are read to determine the number and types of each simple load defined in the bulk data packet. The **PG** matrix is flushed and initialized prior to the start of the loads assembly loop. This loop consists of a search to determine if the load case is

(1) a simple mechanical load

(2) a simple gravity load

(3) a simple thermal load

(4) a combination of mechanical and/or gravity loads

The column of the **PG** matrix associated with each right-hand side is assembled using the **SMPLOD** (and **LOAD**) data. The thermal and gravity loads are special in that the **GLBDES** information must be retrieved in order to assemble the loads representing the current design. The case where no design variables are defined does not represent a special case, however, since the **DPVRGI** and **DPTHGI** entities always include terms representing the "zeroth" design variable. Once all the **STATICS** cases have been processed, the module terminates.

Design Requirements:

1. The module assumes that at least one **STATIC** load case is defined in the **CASE** relation for the current boundary condition.

2. The **SMPLOD** entity from the **LODGEN** module must exist as must the **DPVRGI** and **DPTHGI** gravity and thermal load sensitivity matrices.

Error Conditions:

1. No simple loads are defined in the **SMPLOD** entity

Engineering Application Module: **IFP**

Entry Point: **IFP**

Purpose:

To process the Bulk Data Pocket and to load the input data to the data base. Also, to compute the external coordinate system transformation matrices and to create the basic grid point data.

MAPOL Calling Sequence:

```
CALL IFP ( GSIZEB, EIDTYPE );
```

GSIZEB	The size of the structural set (Integer, Output)
EIDTYPE	Relation containing element identification numbers and their corresponding element type (Character,Output)

Application Calling Sequence:

None

Method:

The Input File Processor module performs several tasks to initialize the execution of ASTROS procedure. It begins by setting the titling information for the **IFP** bulk data echo (should that option be selected).

Following these tasks, the module continues with the interpretation of the bulk data packet of the input stream. This packet resides on an unstructured entity called **&BKDTPKT** which is loaded by the executive routine **PREPAS** during the interpretation of the input data stream. The **IFP** module proceeds in two phases. In the first phase, the bulk data are read, expanded from free to fixed format and sorted on the first three fields of each bulk data entry. If an unsorted echo is requested, that echo is performed as the **&BKDTPKT** entity is read. If a sorted echo is desired, it is performed after the expansion and sort has taken place. In either case, the bulk data is sorted by the **IFP** module. The resultant data are stored on one or more scratch unstructured entities depending on how many passes must be performed to accomplish the sort in the available memory. If all the bulk data fits into open core, only a single scratch file is required.

For the **MODEL** punch option request, the expanded and sorted input Bulk Data entries are divided into following categories:

(1) element definition entries (e.g. **CBAR**)

(2) property definition entries (e.g. **PBAR**)

(3) design variable linking and design constraint definition entries (e.g. **DESELM, DESVARP, DESVARS** and **DCONVMM, DCONxxxx**)

(4) the rest of the Bulk Data

Those entries in categories (1), (2) and (4) are stored in corresponding unstructured entities for use in module **DESPUNCH**. Those in category (3) are not saved for **DESPUNCH**, since it will output a **MODEL** without the design entries.

The second phase of the bulk data interpretation proceeds based on the sorted bulk data from the expansion phase. This phase begins by reading the first bulk data entry in the sorted list and locating its bulk data template in the set of templates stored on the system data base by the SYSGEN program. This template defines the card field labels, the field variable type, the field default value, the field error checks and information on where to load the field into the data base loading array. The template is

compiled once and all like bulk data entries are processed together. Any user input errors that are detected are flagged with a message indicating the field that is in error and whether the error consists of an illegal data type (i.e, an integer value in a real field) or of an illegal value for the given field (i.e., a negative element identification number). Note that the **IFP** module is only checking errors on a single bulk data entry and does not perform any inter-entry compatibility checks.

This process is then repeated for each different bulk data entry type in the sorted list of bulk data entries. If any errors have occured, the module terminates the ASTROS execution. As a final two steps, the **IFP** module performs calls to the **MKTMAT**, **MKBGPD** and **MKUSET** submodules to create the transformation matrices for any external coordinate systems, to generate the basic grid point data and to make an error checking pass over the structural set definitions. These three tasks are not explicitly part of the **IFP** module but are so basic to every execution that they cannot properly be considered MAPOL engineering application modules. Any errors resulting in these two tasks will also cause the run to terminate with the appropriate error messages.

Design Requirements:

None

Error Conditions:

1. User bulk data errors are flagged and cause program termination.
2. Inconsistent or illegal coordinate system definitions.
3. Illegal grid/scalar and/or extra point definitions.
4. Illegal structural set definitions in the **MODEL**.

Engineering Application Module: **INERTIA**

Entry Point: **INRTIA**

Purpose:

To compute the rigid body accelerations for statics analyses with inertia relief.

MAPOL Calling Sequence:

```
CALL INERTIA ( [LHS(BC)], [RHS(BC)], [AR] );
```

[LHS(BC)]	Rigid body reduced mass matrix (Input), where **BC** represents the MAPOL boundary condition loop index number
[RHS(BC)]	Applied load vector reduced to the r-set (Input), where **BC** represents the MAPOL boundary condition loop index number
[AR]	Matrix of acceleration vectors (Output)

Application Calling Sequence:

None

Method:

Matrices **LHS** and **RHS** are read into memory and **AR** is computed by solving **[LHS] [AR] = [RHS]**

Design Requirements:

1. There must be an r-set and the reductions to the r-set must have been performed.

Error Conditions:

1. The **LHS** matrix is singular.

Engineering Application Module: **ITERINIT**

Entry Point: **ITINIT**

Purpose:

Initializes the **CONST** relation for the current iteration.

MAPOL Calling Sequence:

```
CALL ITERINIT ( NITER, CONST, LAMBDA );
```

NITER	Design iteration number (Integer, Input)
CONST	Relation of design constraints (Input and Output)
LAMBDA	Relation of extracted eigenvalues (Character, Input and Output)

Application Calling Sequence:

None

Method:

This module ***must*** be called at the top of each design iteration loop. Its function is twofold: 1) to set the iteration number page header into **SUBTITLE(88:)** in the **/OUTPT2/** common and 2) to reset the **CONST** relation on restart runs.

Each page of output during the design iterations is labeled in the **SUBTITLE** line with the iteration number. It is this module that sets that part of the **SUBTITLE** line that contains that information.

The **CONST** relation is opened and, if not empty, a conditioned retrieval is done to see if any entries exist with an iteration number greater than or equal to the current **NITER**. If so, the **CONST** relation entries with **NITER** values less than the current **NITER** are copied to a scratch **CONST** relation, the scratch name is exchanged for the old **CONST** name and the scratch entity (now pointing to the original **CONST**) is destroyed. Thus, all entries with **NITER** values associated with iterations that have not yet occurred are flushed. This resetting of the **CONST** relation is done so that ASTROS can be restarted at a previous design iteration merely by setting the value of **NITER** in the MAPOL sequence back to the desired starting iteration number.

If **CONST** is empty or if no restart actions are required, **CONST** is closed and the module terminates without action.

Design Requirements:

1. **ITERINIT** is one of the few application modules that is allowed to touch the **TITLE**, **SUBTITLE** and **LABEL** entries of the **/OUTPT2/** common beyond the 72nd character. While applications may set the first 72 characters with the input **TITLE**, etc., generally only the system may modify them beyond that. These labels are used by **UTPAGE**.

Error Conditions:

None

Engineering Application Module: **LAMINCON**

Entry Point: **LAMCON**

Purpose:

To evaluate composite laminate constraints defined on **DCONLAM**, **DCONLMN** and **DCONPMN** bulk data entries.

MAPOL Calling Sequence:

```
CALL LAMINCON ( NITER, NDV, DCONLAM, DCONLMN, DCONPMN, TFIXED, GLBDES,
                LOCLVAR, [PTRANS], CONST );
```

NITER	Design iteration number (Integer, Input)
NDV	The number of global design variables (Integer, Input)
DCONLAM	The relation containing the **DCONLAM** entries (Input)
DCONLMN	The relation containing the **DCONLMN** entries (Input)
DCONPMN	The relation containing the **DCONPMN** entries (Input)
TFIXED	Relation of fixed thicknesses of undesigned layers of designed composite elements (Output)
GLBDES	Relation of global design variables (Input)
LOCLVAR	Relation containing the relationship between local variables and global variables in the design problem (Input)
[PTRANS]	The design variable linking matrix (Input)
CONST	Relation of constraint values (Output)

Application Calling Sequence:

None

Method:

The **LAMINCON** module begins by checking if the **DCONxxxx** entities contain any entries. If there are any entries, they are considered to be design constraints and are imposed (computed). To set up for the computations, the local design variable data, **ELEMLIST** and **PLYLIST** data are read into memory. Then each type of constraint is processed in the order: ply minimum gauge, laminate minimum gauge and laminate composition. As each constraint is computed, it is stored to the **CONST** relation. The **TFIXED** relation contains the thicknesses of all undesigned layers of composite elements and is used in the evaluation of these constraints to determine the thicknesses of layers defining either the *ply* or the *laminate*.

Design Requirements:

None

Error Conditions:

1. Missing **PLYLIST** or **ELEMLIST** data referenced on **DCONxxxx** entries
2. Ply or laminate definitions that include only undesigned layers.

Engineering Application Module: **LAMINSNS**

Entry Point: **LAMSNS**

Purpose:

To evaluate the sensitivities of composite laminate constraints defined on **DCONLAM**, **DCONLMN** and **DCONPMN** bulk data entries.

MAPOL Calling Sequence:

```
CALL LAMINSNS ( NITER, NDV, GLBDES, LOCLVAR, [PTRANS], CONST, [AMAT] );
```

NITER	Design iteration number (Integer, Input)
NDV	The number of global design variables (Integer, Input)
GLBDES	Relation of global design variables (Input)
LOCLVAR	Relation containing the relationship between local variables and global variables in the design problem (Input)
[PTRANS]	The design variable linking matrix (Input)
CONST	Relation of constraint values (Input)
[AMAT]	Matrix of constraint sensitivities (Output)

Application Calling Sequence:

None

Method:

The **LAMINSNS** module begins by checking the **CONST** relation to see if any of the active constraints are **DCONLAM**, **DCONPMN** or **DCONLMN**. These constraint types are processed in this module if any are active.

If any active laminate constraints are present, their sensitivities are computed from the data in the **CONST** relation and the **[PTRANS]** matrix of sensitivities. The format of the **LOCLVAR** and **[PTRANS]** data are such that, for each row in **LOCLVAR**, the corresponding column in **[PTRANS]** is the sensitivity of the local design variable.

$$P_{ij} = \frac{\partial t_i}{\partial v_j}$$

These are the constituents of the derivative computations. To compute each of the constraint derivatives, the appropriate columns, $\frac{\partial t_i}{\partial v}$, are summed and combined with scale factors such as the current thickness and allowable value as shown below.

For ply and laminate minimum gauges, the constraint derivative is computed as:

$$\frac{\partial g}{\partial v} = -\frac{1}{t_{\min}} \sum_{i=1}^{nply} \frac{\partial t_i}{\partial v}$$

where

t_{min} = *ply* or *laminate* minimum gauge

nply = number of *designed* plies defining the ply or laminate

For laminate composition constraints, the constraint derivatives are different depending on whether an upper or lower bound constraint is imposed:

$$\frac{\partial g_{upper}}{\partial v} = \frac{1}{t_{lam}^2}\left[t_{lam} \sum_{i=1}^{npp} \frac{\partial t_{ply_i}}{\partial v} - t_{ply} \sum_{j=1}^{npl} \frac{\partial t_{lam_j}}{\partial v} \right]$$

$$\frac{\partial g_{lower}}{\partial v} = \frac{1}{t_{lam}^2}\left[t_{lam} \sum_{j=1}^{npl} \frac{\partial t_{lam_j}}{\partial v} - t_{ply} \sum_{i=1}^{npp} \frac{\partial t_{ply_i}}{\partial v} \right]$$

where

t_{lam} = current *laminate* thickness

t_{ply} = current *ply* thickness

npp = number of layers in the current *ply*

npl = number of layers in the current *laminate*

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **LODGEN**

Entry Point: **LODGEN**

Purpose:

To assemble the linearly designed simple load vectors and linear simple load sensitivities for all applied loads in the Bulk Data packet.

MAPOL Calling Sequence:

```
CALL LODGEN ( GSIZEB, GLBDES, DVCT, DVSIZE, GMMCT0, DMVI0, TELM, TREF, SMPLOD,
              [DPTHVI], [DPGRVI] );
```

GSIZEB	Length of the g-set vectors (Integer, Input)
GLBDES	Relation of global design variables (Input)
DVCT	Relation containing the data required for the assembly of the linear design sensitivity matrices (Input)
DVSIZE	Unstructured entity containing memory allocation information on the **DVCT** relation (Input)
GMMCT0	Relation containing connectivity data for the **DMVI0** linear sensitivity matrix (Input)
DMVI0	Unstructured entity containing the linear mass design sensitivity
TELM	Unstructured entity containing the linear design variable element thermal load partitions (Input)
TREF	Unstructured entity containing the element reference temperature (Input)
SMPLOD	Unstructured entity of simple load vector information (Output)
[DPTHVI]	Matrix entity containing the linear thermal load sensitivities (Output)
[DPGRVI]	Matrix entity containing the linear gravity load sensitivities (Output)

Application Calling Sequence:

None

Method:

The module begins with a call to subroutine **LDCHK** which performs extensive error checking on the bulk data and solution control commands related to applied loads and performs bookkeeping tasks prior to the computation of the simple loads. Control is then returned to **LODGEN** and **CSTM** and **BGPDT** data are read into core. A loop on the number of unique external load ID's is then begun. Calls to **PCONST** and **PFOLOW** place mechanical loads bulk data information into a **GSIZE** loads vector. This vector is then written to the **SMPLOD** unstructured entity and the process is repeated for the remaining external loads. If there are thermal loads, a call to **THRMLS/D** creates columns of the **DPTHGI** matrix based on linear design element thermal matrices and temperature data. If there are gravity loads, a call to **GRAVTS/D** constructs acceleration vectors and then computes **DPVRGI** columns based on the acceleration vectors and the **DMVI0** unstructured entity. The **DVCT**, **TELM** and **TREF** entities are purged and control is returned to the executive.

Design Requirements:

1. For the general case, this should be the last preface module because it may require inputs from **EMG** and **EMA1**.

Error Conditions:

None

Engineering Application Module: **MAKDFU**

Entry Point: **MAKDFU**

Purpose:

To assemble the sensitivities to the displacements of active stress and displacement constraints in the current active boundary condition.

MAPOL Calling Sequence:

```
CALL MAKDFU ( NITER, BC, GSIZEB, [SMAT], [NLSMAT], SMATCOL, NLSMTCOL,
              [GLBSIG], [NLGLBSIG], CONST, BGPDT, [DFDU], ACTUAGG, SUB );
```

NITER	Design iteration number (Integer, Input)
BC	Boundary condition identification number (Integer, Input)
GSIZEB	The size of the structural set (Integer, Input)
[SMAT]	Matrix entity containing the linear portion of the sensitivity of the stress and strain components to the global displacements (Input)
[NLSMAT]	Matrix entity containing the nonlinear portion of the sensitivity of the stress and strain components to the global displacements (Input)
SMATCOL	Relation containing matrix **SMAT** column information (Character,Input)
NLSMTCOL	Relation containing matrix **NLSMAT** column information (Character,Input)
[GLBSIG]	Matrix of stress/strain components for all the applied linearly designed stress constraints for the current boundary condition (Input)
[NLGLBSIG]	Matrix of stress/strain components for all the applied nonlinearly designed stress constraints for the current boundary condition (Input)
CONST	Relation of constraint values (Input)
BGPDT(BC)	Relation of basic grid point coordinates (Character,Input)
[DFDU]	Matrix containing the sensitivities of active displacement and/or stress-strain constraints to the displacements (Output)
ACTUAGG	Logical flag to indicate whether any **DFDU** terms exist (Logical, Output)
SUB	An optional flag which indicates whether statics or static aeroelasticity is associated with the constraints in this call. The discipline flag = **0** if **STATICS** = subscript identifier, **SUB**, of the aeroelastic subcases if **SAERO** (Integer, Input)

Application Calling Sequence:

None

Method:

For the current active boundary condition, the **MAKDFU** module begins by processing the active displacement constraints. The **CONST** relation is queried for all active displacement constraints (**CTYPE=3**). Each tuple that qualifies the active condition is processed using the **PNUM** attribute to

position to the appropriate location within the **DCENT** entity. The **DCENT** terms are loaded in the **DFDU** matrix in the order that active displacement constraints are encountered in the **CONST** relation. Constraints are evaluated for each load condition within the active boundary condition in constraint type order. The **DFDU** matrix is thus also formed in this order but the inactive constraints are ignored.

After processing the active displacement constraints (if any), the **MAKDFU** module processes the active stress/strain constraints. The **CONST** relation is conditioned to retrieve the active stress and/or principal strain constraints (**CTYPE**'s **4**, **5** and **6**). For each active constraint, the current boundary condition number and the load condition number (stored on the **CONST** relation in the **SCEVAL** module) are used to determine the column number of the **SMAT** or **NLSMAT** matrix that holds the sensitivity of the current stress term to the displacements. Having recovered the **SMAT** or NLSMAT columns for the current active constraint, the **DFDU** terms are computed based on the element type and constraint type. Where the sensitivity is a function of the stress/strain values, the appropriate rows of the **GLBSIG** or **NLGLBSIG** column associated with the current boundary condition/load condition/discipline are retrieved for use in the computations.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **MAKDFV**

Entry Point: **MAKDFV**

Purpose:

To assemble the sensitivities of active thickness constraints.

MAPOL Calling Sequence:

```
CALL MAKDFV ( NITER, NDV, [PMINT], [PMAXT], CONST, GLBDES, DESLINK,
              FDSTEP, [AMAT] );
```

NITER	Design iteration number (Integer, Input)
NDV	The number of global design variables (Integer, Input)
[PMINT]	Matrix entity containing the minimum thickness constraint sensitivities (Input)
[PMAXT]	Matrix entity containing the maximum thickness constraint sensitivities (Input)
CONST	Relation of constraint values (Input)
GLBDES	Relation of global design variables (Character,Input)
DESLINK	Relation of design variable linking information (Character,Input)
FDSTEP	Relative design variable increment for finite difference (Real,Input)
[AMAT]	The matrix of constraint sensitivities to the global design variables (Output)

Application Calling Sequence:

None

Method:

The **MAKDFV** module begins by determining if any active thickness constraints exist for this design iteration. The **CONST** relation is conditioned to retrieve active minimum and maximum thickness constraints. If any active constraints are found, they are processed in the order recovered from the **CONST** relation; that is, active minimum thickness constraints followed by active maximum thickness constraints. Since the constraint sensitivities are functions of the current local variable value when they are controlled by move limits rather than gauge limits, the execution of the module proceeds with the calculation of all the individual layer thicknesses for all the elements designed by shape functions. Since move limits are considered to be desirable in the vast majority of cases and because there is no reliable way to determine before-hand if any particular active constraint is move limit controlled, the local variables are always computed in this module. The **PTRANS** matrix, prepared in the **MAKEST** module is used to evaluate these thicknesses:

$$\{t\} = [PTRANS]^T\{v\}$$

After the local variables have been computed, the **LOCLVAR** relation (also built in the **MAKEST** module) is used to determine the current total thickness for a layered composite element. The **VFIXED** entity gives that portion of the thickness of composite elements that is not designed. The sensitivities of the thickness constraints are essentially the appropriate column of the **PMINT** or **PMAXT** matrix. The column is identified by the **PNUM** attribute of the **CONST** relation. If the particular local variable constraint is

controlled by move limits, however, the sensitivity becomes a function of the current thickness and must be adjusted accordingly. This applies only to minimum gauge constraints, however, since move limits are not applied to maximum thickness constraints. The resulting constraint sensitivities are loaded, in the order processed, onto the **AMAT** matrix.

Design Requirements:

1. The move limit that is passed into this routine ***must*** match the value used to evaluate the constraints in the **TCEVAL** module. If not, the constraint sensitivities will be in error with no warning given.

Error Conditions:

1. The move limit must be greater than 1.0 if it is imposed.

Engineering Application Module: **MAKDVU**

Entry Point: **MAKDVU**

Purpose:

To multiply the stiffness or mass design sensitivities by the active displacements or accelerations.

MAPOL Calling Sequence:

```
CALL MAKDVU ( NITER, NDV, GLBDES, [UGA], [DKUG], GMKCT, DKVI );
```

NITER	Design iteration number (Integer, Input)
NDV	Number of design variables (Integer, Input)
GLBDES	Relation of global design variables (Input)
[UGA]	Matrix of "active" displacements or accelerations for the current boundary condition (Input)
[DKUG]	The product of the design sensitivity matrices and the active displacement/acceleration vectors (Output)
GMKCT	Relation containing connectivity data for the **DKVI** sensitivity matrix (Input)
DKVI	Unstructured entity containing the stiffness or mass design sensitivity matrix in a highly compressed format (Input)

Application Calling Sequence:

None

Method:

This is a utility module that performs a matrix multiplication of a g-set matrix of displacements or accelerations and the g-set sized design sensitivities **DKVI** or **DMVI** entities that are the **NDV** g x g design sensitivity matrices stored in a highly compressed format.

The module first reads in design variable information (**ID** and value) and then space is reserved for the maximum **DKVI** record. A determination is made as to how many columns of **UGA** and **DKUG** can be held in core simultaneously. Spill logic is used if not all the columns can be processed simultaneously. Columns of **UGA** are read into core and a loop on the number of design variables is made to calculate the columns of the **DKUG** matrix. Care is taken to write null columns when a particular design variable has no **DKVI** entries. The **UNMTML** subroutine is called by **MAKDVU** to multiply the unstructured data and the response vector.

Design Requirements:

1. The format of the **DKVI/GMKCT** inputs is assumed to parallel the structure of those entities output from **EMA1**.

Error Conditions:

None

Engineering Application Module: **MAKEST**

Entry Point: **MAKEST**

Purpose:

To generate the element summary entities for all structural elements. Also, to determine the design variable linking and generate sensitivities for any thickness constraints.

MAPOL Calling Sequence:

```
CALL MAKEST ( NDV, GLBDES, [PTRANS], [PMINT], [PMAXT], LOCLVAR,
              TFIXED, DESLINK );
```

NDV	The number of global design variables (Integer, Output)
GLBDES	Relation of global design variables (Output)
[PTRANS]	The design variable linking matrix (Output)
[PMINT]	Matrix entity containing the minimum thickness constraint sensitivities (Output)
[PMAXT]	Matrix entity containing the maximum thickness constraint sensitivities (Output)
LOCLVAR	Relation containing the relationship between local variables and global variables in the design problem (Output)
TFIXED	Relation of fixed thicknesses of undesigned layers of designed composite elements (Output)
DESLINK	Relation of design variable connectivity from **MAKEST** module containing one record for each global design variable connected to each local variable. (Output)

Application Calling Sequence:

None

Method:

The **MAKEST** module performs the first phase of the structural element preface operations with the **EMG** and **NLEMG** module performing the second phase. The first action of the module is to perform the uniqueness error checks on the element bulk data as stored on the data base by the **IFP** module. These checks ensure that all property entries have unique identification numbers within each property type (with the exception of the **PCOMPi** entries where duplicate ID's signify different composite layers). Also, unique identification numbers for the **MATi** entries are enforced across all **MATi** types. The **MAKEST** module then performs the initial processing of the design variable linking in the **PREDES** module. The **GLBDES** relation is set up in memory with several columns to be filled in as the design variable linking is continued later in the module. If there are design variables defined in the bulk data, the number of global design variables, **NDV**, is determined for output to the MAPOL sequence and a number of scratch and hidden entities are opened to prepare for the design variable linking task performed in this module.

The **MAKEST** module continues by reading in the **BGPDT** data and initializing the **PTRANS**, **PMINT**, and **PMAXT** matrix columns that are built on the fly in the element dependent routines. The module then calls each element dependent routine in turn. The order in which these submodules are called is very important in that it provides an implicit order for the **MAKEST**, **EMG**, **NLEMG**, **SCEVAL**, **EDR** and **OFP** modules.

That order is alphabetical by connectivity bulk data entry and results in the following sequence:

(1) Bar elements

(2) Scalar spring elements

(3) Linear isoparametric hexahedral elements

(4) Quadratic isoparametric hexahedral elements

(5) Cubic isoparametric hexahedral elements

(6) Scalar mass elements

(7) General concentrated mass elements

(8) Rigid body form of the concentrated mass elements

(9) Isoparametric quadrilateral membrane elements

(10) Quadrilateral bending plate elements

(11) Rod elements

(12) Shear panels

(13) Triangular bending plate elements

(14) Triangular membrane elements

Within each element dependent routine, the **xxxEST** relation for the element is opened and flushed. If design variables exist in the **MODEL**, the **ELIST**, **PLIST** and **SHAPE** entries associated with this element type (if the element can be designed) are opened and read into memory for use in the design variable linking. Then the connectivity relation for the element is opened and the main processing loop begins. Each tuple is read, the grid point references are resolved into internal sequence numbers and coordinates, the property entry is found from the proper property relation(s) and the **EST** relation tuple is formed in memory. Numerous checks on the existence of grid points, property entries and the uniqueness of the element identification number within each element type are performed.

Finally, if there are design variables, the **DESCHK** submodule is called to determine whether the element is linked to a design variable. The **DESCHK** utility searches the in-core **GLBDES**, **ELIST**, **PLIST** and/or **SHAPE** data and determines if the current element is designed. Also, the final attributes of the **GLBDES** relation for physical and shape function linking are completed. The module performs error checks to ensure that the rules for design variable linking are satisfied for each particular global design variable and element.

On return to the element **EST** routine, the **LOCLVAR**, **PTRANS**, **PMINT** and/or **PMAXT** entities are built for the local design variable if the element was found by **DESCHK** to be designed. Finally, the constraint flags, design flags, design variable nonlinear flag, composite type flag and thermal stress information are set. The constraint and thermal stress attributes will be revised as needed in the **EMG** and **NLEMG** module.

When all the elements have been processed, the **EST** relation for the element type is loaded to the data base. Care is taken that the final relation is sorted by the element identification number. When all the element routines have been called, the **DESLINK** entity, which was formed on the fly in the element routines, is loaded to the data base. This entity contains the number of and identification numbers for each design variable connected to each designed element. These data are used to generate the **DVCT**, **DVCTD** and/or **DDVCT** relations in the **EMG** and NLEMG module. All the other design variable linking entities that have been built on the fly are also closed. Any queued error messages are dumped to the user file and the module terminates.

Design Requirements:

1. The basic connectivity data from the **IFP** module must be available.

Error Conditions:

1. Numerous error checks are performed on the consistency of the bulk data for structural element definition as well as of element geometry and connectivity.

2. Design variable linking errors are flagged.

Engineering Application Module: **MK2GG**

Entry Point: **MK2GG**

Purpose:

Interprets case control for the current boundary condition and outputs the **M2GG** and/or **K2GG** matrices if any.

MAPOL Calling Sequence:

```
CALL MK2GG ( BC, GSIZEB, [M2GG], M2GGFLAG, [K2GG], K2GGFLAG );
```

BC	Boundary condition identification number (Integer, Input)
GSIZEB	Number of g-set DOF's excluding any that may have been added on earlier iterations by **GDR** (Integer, Input)
[M2GG]	Direct input g-set mass matrix for the current **BC** (Optional, Output)
M2GGFLAG	Flag indicating whether **M2GG** was loaded with data (Optional, Logical, Output)
[K2GG]	Direct input g-set stiffness matrix for the current **BC** (Optional, Output)
K2GGFLAG	Flag indicating whether **K2GG** was loaded with data (Optional, Logical, Output)

Application Calling Sequence:

None

Method:

First the **CASE** relation is read for the current boundary condition to determine if **M2GG** or **K2GG** matrices were named. Error checking is performed to ensure that an output matrix is passed to **MK2GG** for both matrices if both are named in **CASE**. The arguments are otherwise optional. Further, the entities named in **CASE** are checked to ensure that they are matrices and that they are square and of the proper row and column dimensions (**GSIZEB** x **GSIZEB**).

Then the named output matrix is created, or if it already exists, flushed. The **APPEND** utility is used to copy the named entity onto the output entity.

Design Requirements:

1. The **DMIG** or **DMI** entries that may be sources of the **M2GG** and/or **K2GG** matrices must be processed prior the the calling of this module. This module assumes that the named entities already exist.

Error Conditions:

1. **x2GG** entities do not exist.
2. **x2GG** entities are not matrix entities
3. **x2GG** entities are not of the proper dimension.
4. All errors cause ASTROS termination.

Engineering Application Module: **MKAMAT**

Entry Point: **MKAMAT**

Purpose:

To assemble the constraint sensitivity matrix from the sensitivity matrices formed by **MAKDFU** and the sensitivities of the displacements for active static load conditions in the current active boundary condition.

MAPOL Calling Sequence:

```
CALL MKAMAT ( [AMAT], [FIRST], [SECOND], [THIRD], PCA, PRA, [PGA] );
```

[AMAT]	Matrix of sensitivities of the constraints to the design variables (Input and Output)
[FIRST]	Leading matrix in the multiplication to obtain **AMAT** (Input)
[SECOND]	Trailing matrix in the multiplication to obtain **AMAT** (Input)
[THIRD]	The matrix to be added is the multiplication to obtain **AMAT** (Input)
PCA	Unstructured entity which contains the unique subcase numbers for the constraints that are active for the boundary condition. Only constraints for the current boundary condition are included in the list (Input)
PRA	Unstructured entity which contains the unique subcase numbers for the displacement and/or element stress/strain response functions that are required by active user function constraints (Character,Input)
[PGA]	Partition vector for active displacement vectors (Input)

Application Calling Sequence:

None

Method:

Conceptually, the module multiplies the transpose of the **FIRST** matrix times the **SECOND** and adds the **THIRD**. The data in the three matrices are determined based on whether the gradient method or the virtual loads method of sensitivity analysis is being employed (see Subsection 6.3 of the Theoretical Manual). The matrix multiplication is complicated by the fact that it may be necessary to partition the matrices for each subcase that is active in the boundary condition.

The module begins by reading the **PCA** and **PGA** information into core. The number and identity of the active subcases is determined. If the number is greater than one, thirteen scratch matrix entities are created to store intermediate data. A loop on the number of active subcases then occurs. If it is not the last pass through this loop, the **FIRST** matrix is partitioned to obtain the **NDV** columns that apply for the current subcase, and the **SECOND** matrix is partitioned to obtain only the columns that correspond to active constraints for the subcase and the **THIRD** matrix is partitioned to obtain the **NDV** rows that apply for the current subcase and the columns that correspond to active constraints for the subcase.

PRA and **PGA** are used to partition the **FIRST**, **SECOND** and **THIRD** matrices to obtain the displacement and/or element stress and strain response sensitivities which are required by active user function constraints. Those sensitivities are stored into relation and matrix entities to be used by user function evaluation utilities.

The algorithm is somewhat more complicated than this in that the parts of the matrices that remain after partitioning are renamed to **FIRST** and **SECOND** so that the partitioning operation becomes successively smaller and no partition is required on the last pass through the loop. The extracted matrices are then multiplied and the resulting matrix is either **AMAT** (when there is only one active subcase and the **AMAT** matrix was empty on entering the module) or it is appended to **AMAT**. Once the loop is completed, any scratch matrices are destroyed and control is returned to the executive.

Design Requirements:

1. This module is invoked at the end of the boundary condition loop in the sensitivity analysis portion of the MAPOL sequence.

2. It is called only if there are active stress and displacement constraints for the boundary condition.

Error Conditions:

None

Engineering Application Module: **MKDFDV**

Entry Point: **MKDFDV**

Purpose:

Computes the sensitivity of BAR element cross-sectional dimension relation constraints to design variables

MAPOL Calling Sequence:

```
CALL MKDFDV ( NITER, NDV, CONST, DESLINK, GLBDES, [AMAT] );
```

NITER	Design iteration number (Integer,Input)
NDV	Number of design variables (Integer,Input)
CONST	Relation of constraint values (Character,Output)
DESLINK	Relation of design variable linking information (Character,Input)
GLBDES	Relation of global design variables (Character,Input)
[AMAT]	Matrix of sensitivities of constraints to the design variables (Character,Output)

Application Calling Sequence:

None

Method:

This module first gets all active BAR element cross-sectional dimension relation constraints. The sensitivities of each active constraint to the design variables are computed by multiplying the sensitivities of the constraint to the dimensions (factors in relation **CONST**) with the sensitivities of the dimensions to the design variables (**PREF** in **DESLINK**). Those sensitivities are then stored in columns of matrix **[AMAT]**.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **MKDFSV**

Entry Point: **MKDFSV**

Purpose:

To calculate matrix **[DFSV]** which contains the S-matrix derivatives related to active stress/strain constraints. The stress/strain constraints are functions of the product of the S-matrix and the displacements:

$$g = f\ (\boldsymbol{Su})$$

The sensitivities of these constraints to the designed variables is decomposed into two parts. The first is a function of the product of the S-matrix derivatives and displacements, and the second is a function of the product of the S-matrix and the displacement derivatives:

$$\frac{\partial g}{\partial v} = \bar{f}\left(\frac{\partial \boldsymbol{S}}{\partial v}\boldsymbol{u}\right) + h\left(\boldsymbol{S}\frac{\partial \boldsymbol{u}}{\partial v}\right)$$

The **DFSV** matrix represents the first part:

$$\mathbf{DFSV} = f\left(\frac{\partial \boldsymbol{S}}{\partial v}\boldsymbol{u}\right)$$

MAPOL Calling Sequence:

```
CALL MKDFSV ( NITER, BC, GSIZEB, [NLGLBSIG], CONST, [NLSMAT], NLSMTCOL,
              [UGA], DESLINK,DSCFLG, NDV, GLBDES, LOCLVAR, [PTRANS],
              [DFSV], DELTA );
```

NITER	Optimization iteration number (Integer, Input)
BC	The MAPOL boundary condition loop index number (Integer, Input)
GSIZEB	number of dofs in the structural set (Integer, Input)
[NLGLBSIG]	Stress vectors for design variable nonlinearly constrained elements (Character, Input)
CONST	Relation of constraints (Character, Input)
[NLSMAT]	Matrix entity containing the nonlinear portion of the sensitivity of the stress and strain components to the global displacements (Character,Input)
NLSMTCOL	Relation containing matrix **NLSMAT** column information (Character,Input)
[UGA]	Active displacement vectors at current boundary condition (Character, Input)
DESLINK	Relation of design variable linking
DSCFLG	Discipline flag (Integer, Input) = 0 statics ≠ 0 static aeroelasticity
NDV	Number of design variables (Integer, Input)
GLBDES	Global design variable relation (Character, Input)

LOCLVAR	Local design variable relation (Character, Input)
[PTRANS]	Design variable linking matrix (Character, Input)
[DFSV]	Matrix contains S-matrix derivatives related active stress/strain constraints (Character, Output)
DELTA	The relative design variable increment for finite difference computation. (Real, Input)

Application Calling Sequence:

None

Method:

This module first gets the **DVID** list from **GLBDES**. The module then gets **EST** entries for nonlinearly designed constraint **QUAD4** and **TRIA3** elements and places them into incore lists. Then the module determines the number of active displacement constraints, and gets active stress and strain constraints for this design iteration. Null columns are stored in **DFSV** corresponding to active displacement constraints so that **DFSV** will be compatible in module **MKAMAT**. For each active stress/strain constraint, the following operations are applied. If it is a linearly designed constraint, a null column is stored in **DFSV**, otherwise, matrix **DSDT** which contains the sensitivities of the nonlinear S-matrix to the related local design variables is computed for the element related to this constraint. Matrix **DSDV** which contains the sensitivities of the nonlinear S-matrix to the related global design variables is computed by using the **DSDT** matrix and design variable linking factors from **DESLINK**. Matrix **DSVU** is the multiplication of the transposed active displacement vector **UGA** times **DSDV**. The **DFSV** term at the row number corresponding to that active constraint is computed from **DSVU**, the constraint value, the related stress/strain values from **NLGLBSIG**, and the constraint allowables.

Design Requirements:

1. This module must be called prior to **MKAMAT** and after the active displacement vector is available.

Error Conditions:

None

Engineering Application Module: **MKPVECT**

Entry Point: **PVCDRV**

Purpose:

To generate partitioning vectors from unstructured entity **USET**.

MAPOL Calling Sequence:

```
CALL MKPVECT ( USET(BC), [PGMN(BC)], [PNSF(BC)], [PFOA(BC)], [PARL(BC)] );
```

USET(BC)	Unstructured entity defining structural sets for each degree of freedom (Character,Input), where **BC** represents the MAPOL boundary condition loop index number
[PGMN(BC)]	The vector partitioning the structural degrees of freedom into the independent and the multipoint constraint degrees of freedom (Character, Output), where **BC** represents the MAPOL boundary condition loop index number
[PNSF(BC)]	The vector partitioning the independent degrees of freedom into the free and the single point constraint degrees of freedom (Character, Output), where **BC** represents the MAPOL boundary condition loop index number
[PFOA(BC)]	The vector partitioning the free degrees of freedom into the analysis set and the omitted degrees of freedom (Character, Output), where **BC** represents the MAPOL boundary condition loop index number
[PARL(BC)]	The vector partitioning the analysis set degrees of freedom into the l-set and the support degrees of freedom (Character, Output), where **BC** represents the MAPOL boundary condition loop index number

Application Calling Sequence:

```
CALL PVCDRV ( USET, PGMN, PNSF, PFOA, PARL )
```

Method:

This module first reads the **USET** record which contains the number of degrees of freedom in each dependent set and the bit masks defining the structural sets to which the degrees of freedom belong. Then, for each requested partitioning vector, the bit masks in **USET** are checked for all degrees of freedom with the related structural sets, and then the vector is generated.

Design requirements:

Any of the partitioning vector names may be blank if the corresponding partition will not be used subsequently.

Error:

None

Engineering Application Module: **MKUSET**

Entry Point: **MKUSET**

Purpose:

To generate the structural set definition entity, **USET**, for each boundary condition and to form the partitioning vectors and transformation matrices used in matrix reduction.

MAPOL Calling Sequence:

```
CALL MKUSET ( BCID, GSIZEB, [YS], [TMN], [PGMN], [PNSF], [PFOA],
              [PARL], USET );
```

BCID	User defined boundary condition identification number (Integer, Input)
GSIZEB	The size of the structural set (Integer, Input)
[YS(BC)]	The vector of enforced displacements (Output)
[TMN(BC)]	The transformation matrix for multipoint constraints (Output)
[PGMN(BC)]	The partitioning vector splitting the structural degrees of freedom into the independent and the multipoint constraint degrees of freedom (Output)
[PNSF(BC)]	The partitioning vector splitting the independent degrees of freedom into the free and the single point constraint degrees of freedom (Output)
[PFOA(BC)]	The partitioning vector splitting the free degrees of freedom into the analysis set and the omitted degrees of freedom (Output)
[PARL(BC)]	The partitioning vector splitting the analysis set degrees of freedom into the l-set and the support degrees of freedom (Output)
USET(BC)	The unstructured entity defining structural sets (Output)

Application Calling Sequence:

None

Method:

The **MKUSET** module performs four tasks. The first is to build the **USET** entity of structural set definition masks for the input boundary condition. At the same time, the rigid constraint matrix, **TMN**, relating the dependent multipoint constraint degrees of freedom to the independent degrees of freedom is formed. Also, the vectors of enforced displacements for single point constraints are formed. Lastly, the partitioning vectors for the structural sets are formed.

The generation of boundary condition dependent subscripted matrix entities requires that the **MKUSET** module be called once for each boundary condition in the Solution Control packet. The looping logic is contained in the standard executive sequence rather than within the module itself. Each structural degree of freedom (DOF) is assigned a word in each record of the **USET** entity (aerodynamic degrees of freedom and extra points are ignored). One record is created for each boundary condition in the Solution Control packet. The **MKUSET** module determines to which sets a structural DOF belongs and sets the corresponding bits in the **USET** word associated with that degree of freedom. That word is the bitmask for that degree of freedom.

The assignment of a bit position for each structural set is defined as shown below and are stored in the **/BITPOS/** common block:

SET	BIT POSITION	DESCRIPTION
UX	16	Used for dynamic reduction
UJJP	17	
UJJ	18	
UKK	19	
USB	20	Single point constraints (SPC)
USG	21	Permanent SPCs
UL	22	Free points left for solution
UA	23	Analysis set
UF	24	Free degrees of freedom
UN	25	Independent degrees of freedom
UG	26	Dependent degrees of freedom
UR	27	Support set DOF
UO	28	Omitted (Guyan Reduction) DOF
US	29	Unions of USB and USG sets
UM	30	Dependent MPC DOF

The **MKUSET** module begins by preparing memory blocks for use by the module subroutines. The **BGPDT** tuples associated with structural nodes are brought into core for use in conversion of external identification numbers to internal identification numbers. Each separate structural set is processed by an individual submodule of **MKUSET** with the defaulting for unspecified DOF taking place in the module driver. The **CASE** relation is read to determine the boundary condition definition for the current boundary condition. The submodule **UMSET**, responsible for multipoint constraint set definition also build the **TMN** matrix while the **USSET** submodule for single point constraints builds the **YS** vector. After the **USET** masks have been built for the boundary condition, extensive error checking occurs to ensure that each point is placed in no more than one dependent structural set. If no errors have occured, the **USET** record is written and the associated partitioning vectors are formed.

Design Requirements:

1. The **MKUSET** module requires that the **CASE** relation be complete from the **SOLUTION** module and that the **BGPDT** be formed either by the **BCBGPDT** or **IFP** modules prior to execution.

Error Conditions:

1. Any inconsistent boundary condition specifications are flagged.

2. Any missing bulk data referenced by Solution Control is flagged.

Engineering Application Module: **MSWGRESP**

Entry Point: **MWRESP**

Purpose:

Computes element mass and/or weight intrinsic responses.

MAPOL Calling Sequence:

```
CALL MSWGRESP ( NITER, GLBDES, DESLINK );
```

NITER	Design iteration number (Integer,Input)
GLBDES	Relation of global design variables (Character,Input)
DESLINK	Relation of design variable linking information (Character,Input)

Application Calling Sequence:

None

Method:

This module search through all element weight and mass intrinsic entity entries and obtains all element identification numbers which are required by user function instances. The values of those required element weight and/or mass are obtained from relation **ELMASS** which are generated in module **EMG/NLEMG** and stored into element weight and/or mass intrinsic response entity with instance information.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **NLEMA1**

Entry Point: **NLEMA1**

Purpose:

To assemble the nonlinearly designed element stiffness and mass sensitivity matrix partitions (stored in **DKELM** and **DMELM** entities) with linear stiffness and mass sensitivity matrix partitions (**DKVI0** and **DMVI0**) into the design sensitivity matrices **DKVI** and **DMVI**. To assemble the nonlinear design stiffness and mass matrix partitions (**KELMD** and **MELMD**) with entities **DKVI0** and **DMVI0** into the stiffness and mass matrices **DKVIG** and **DMVIG** for global stiffness and mass matrix assembling.

MAPOL Calling Sequence:

```
CALL NLEMA1 ( NITER, NDV, GLBDES, DVCTD, DDVCT, KELMD, DKELM, MELMD, DMELM,
              GMKCT0, DKVI0, GMMCT0, DMVI0, DWGH1, GMKCT, DKVI, GMMCT,
              DMVI, GMKCTG, DKVIG, GMMCTG, DMVIG, GMMCTD, DMVID, DGMMCT,
              DDMVI, DDWGH2 );
```

NITER	Optimization iteration number (Integer, Input)
NDV	Number of design variables (Integer, Input)
GLBDES	Relation of global design variables (Character, Input)
DCVTD	Relation containing the data required for the assembly of the nonlinear design stiffness and mass matrices (Character, Input)
DDVCT	Relation containing the data required for the assembly of the nonlinear design sensitivity matrices (Character, Input)
KELMD	Unstructured entity containing the nonlinear design stiffness matrix partitions (Character, Input)
DKELM	Unstructured entity containing the nonlinear stiffness sensitivity matrix partitions (Character, Input)
MELMD	Unstructured entity containing the nonlinear design mass matrix partitions (Character, Input)
DMELM	Unstructured entity containing the nonlinear mass sensitivity matrix partitions (Character, Input)
GMKCT0	Relation containing connectivity data for the **DKVI0** linear sensitivity matrix (Character, Input)
DKVI0	Unstructured entity containing the linear stiffness sensitivity matrix in a highly compressed format (Character, Input)
GMMCT0	Relation containing connectivity data for the **DMVI0** linear sensitivity matrix (Character, Input)
DMVI0	Unstructured entity containing the linear mass sensitivity matrix in a highly compressed format (Character, Input)
DWGH1	Unstructured entity containing the linear (invariant) portion of the sensitivity of weight to the design variables (Character, Input)
GMKCT	Relation containing connectivity data for the **DKVI** sensitivity matrix (Character, Output)

DKVI	Unstructured entity containing the total stiffness design sensitivity matrix in a highly compressed format (Character, Output)
GMMCT	Relation containing connectivity data for the **DMVI** sensitivity matrix (Character, Output)
DMVI	Unstructured entity containing the total mass design sensitivity matrix in a highly compressed format (Character, Output)
GMKCTG	Relation containing connectivity data for the **DKVIG** stiffness matrix (Character, Output)
DKVIG	Unstructured entity containing the stiffness matrix in a highly compressed format (Character, Output)
GMMCTG	Relation containing connectivity data for the **DMVIG** mass matrix (Character, Output)
DMVIG	Unstructured entity containing the mass matrix in a highly compressed format (Character, Output)
GMMCTD	Relation containing connectivity data for the **DMVID** mass matrix (Character, Output)
DMVID	Unstructured entity containing the nonlinear part of mass matrix in a highly compressed format (Character,Output)
DGMMCT	Relation containing connectivity data for the **DDMVI** nonlinear mass sensitivity matrix (Character, Output)
DDMVI	Unstructured entity containing the nonlinearly designed mass sensitivity matrix in a highly compressed format (Character, Output)
DDWGH2	Unstructured entity containing the nonlinear portion of the sensitivity of weight to the design variables (Character, Output)

Application Calling Sequence:

None

Method:

The module is executed in two passes; once for nonlinear design stiffness matrices and nonlinear stiffness sensitivity matrices, and a second time for nonlinear design mass matrices and nonlinear mass sensitivity matrices.

In the first pass, **DVCTD** information is read into core one record at a time. The algorithm is structured to maximize the amount of processing done on a given design matrix(typically all of it) in core. Spill logic is in place if a matrix cannot be completely held in core. For the assembly, subroutine **NLRQCR** performs bookkeeping tasks to expedite the assembly and to determine whether spill will be necessary. Subroutine **NLASM1** retrieves **KELMD** information, performs the actual assembly operations and place the results into the **GMKCT8DKVI**, and results in **DGMKCT** and **DDKVI** entities.

If a discipline which requires a mass matrix is included in the solution control, the mass terms are assembled in the second pass. If there are **OPTIMIZE** boundary conditions, this module calculates the nonlinear portion of the sensitivity of the weight to the design variables (**DDWGH2**) and the nonlinear portion of the weight (**DWGH2**) regardless of whether the mass matrices are required. If no mass information is required, the second pass is not made. For the second pass, **MELMD** and **DMELM** data are used. The structure of the assembly operation is otherwise much the same and **GMMCYD, DGMMCY, DMVID** and **DDMVI** data are computed and stored. After those two passes, the total weight is computed from **DWGH1** and **DWGH2. GMKCT0, DKVI0, DGMKCT, DDKVI** are merged into stiffness sensitivity entities **GMKCT**

and **DKVI; GMMCT0, DMVI0, DDGMMCY, DDMVI** are merged into mass sensitivity entities **GMMCT** and **DMVI; GMKCT0, DKVI0, GMMCYD, DKVID** are merged into stiffness matrix entities **GMKCTG** and **DKVIG; GMMCT0, DMVI0, GMMCYD, DMVID** are merged into mass matrix entities **GMMCTG** and **DMVIG.**

Design Requirements:

1. This assembly operation follows **NLEMG** within the MAPOL **OPTIMIZE** iterations.

2. Since gravity loads require **DMVID** and **DDMVI** data, it is necessary to perform **NLEMA1** prior to calling **NLLODGEN. NLEMA1** must always be called before **EMA2.**

Error Conditions:

None.

Engineering Application Module: **NLEMG**

Entry Point: **NLEMG**

Purpose:

To compute the nonlinear design stiffness, mass, thermal load and stress component sensitivities and nonlinear design stiffness and mass matrix partitions.

MAPOL Calling Sequence:

```
CALL NLEMG ( NITER, NDV, GSIZEB, GLBDES, LOCLVAR, [PTRANS], DESLINK,
             [NLSMAT], NLSTMCOL, DVCTD, DDVCT, DVSIZED, DDVSIZE,
             KELMD, DKELM, MELMD, DMELM, TELMD, DTELM, TREFD, FDSTEP );
```

NITER	Optimization iteration number (Integer, Input)
NDV	Number of design variables (Integer, Input)
GSIZEB	The size of the structural set (Integer, Input)
GLBDES	Relation of global design variables (Character, Input)
LOCLVAR	Relation containing the relationship between local variables and global variables in the design problem. (Character, Input)
[PTRANS]	The design variable linking matrix (Character, Input)
DESLINK	Relation of design variable connectivity from **MAKEST** module (Character, Input)
[NLSMAT]	Matrix entity containing the nonlinear portion of the sensitivity of the stress and strain components to the global displacements. (Output)
NLSMTCOL	Relation containing matrix **NLSMAT** column information (Character,Output)
DCVTD	Relation containing the data required for the assembly of the nonlinearly designed stiffness and mass matrices (Character, Output)
DDVCT	Relation containing the data required for the assembly of the nonlinearly designed sensitivity matrices (Character, Output)
DVSIZED	Unstructured entity containing memory allocation information on the **DVCTD** relation. (Character, Output)
DDVSIZE	Unstructured entity containing memory allocation information on the **DDVCT** relation. (Character, Output)
KELMD	Unstructured entity containing the nonlinearly designed stiffness matrix partitions (Character, Output)
DKELM	Unstructured entity containing the nonlinear stiffness sensitivity matrix partitions (Character, Output)
MELMD	Unstructured entity containing the nonlinearly designed mass matrix partitions (Character, Output)
DMELM	Unstructured entity containing the nonlinear mass sensitivity matrix partitions (Character, Output)

TELMD	Unstructured entity containing the nonlinearly designed thermal load partitions (Character, Output)
DTELM	Unstructured entity containing the nonlinear thermal load sensitivity matrix partitions (Character, Output)
TREFD	Unstructured entity containing the element reference temperatures for nonlinearly designed thermal loads. (Character, Output)
FDSTEP	Relative design variable increment for finite difference (Real, Input)

Application Calling Sequence:

None

Method:

The **NLEMG** module performs the nonlinear design variable part of the second phase of the structural element preface operations with the **MAKEST** module performing the first phase.The first action of the **NLEMG** module is to determine if nonlinear design variables and/or thermal loads are defined in the bulk data. If they are, the special actions for design variable linking and thermal stress corrections are taken in the element dependent routines. The **PREMAT** utility to set up the material property data also returns the **SCON** logical flag to denote that there are stress constraints defined in the bulk data. The initialization of the module continues with the retrieval of the **MFORM** data to select lumped or coupled mass matrices in the elements that support both forms. The default is lumped although any **MFORM/COUPLED** (even if **MFORM/LUMPED** also exists) will cause the coupled form to be used. If thermal loads exist, the module prepares the **TREFD** entity to be written by the element dependent routines. The **GLBDES** relation is opened and the design variable identification numbers are read into memory. Finally, the **DDVCT** and **DVCTD** entities are opened and flushed and memory is retrieved to be used in the **NLDVCT** submodule to load the **DDVCT** and **DVCTD** relations. The order in which these submodules are called is alphabetical by connectivity bulk data entry, i.e., (1) Bar elements, (2) Quadrilateral bending plate elements, and (3) Triangular bending plate elements.

Within each element dependent routine, the **xxxEST** relation for the element is opened and read one tuple at a time. If the **EST** relation indicates that the element is nonlinearly designed, the **DESLINK** data is used to write one set of tuples to the **DDVCT** and **DVCTD** relations for each unique design variable linked to the element. The set of tuples consists of one row for each node to which the element is connected. The element dependent geometry processor is then called to generate the **DKELM**, **KELMD**, **DMELM**, **MELMD**, **DTELM** and **TELMD** entries for the element. These data must be generated before the next call to **NLDVCT** since the **DDVCT** and **DVCTD** form the directory to all these entities. Once all the elements are processed within the current element dependent routine, the **TREFD** entity is appended with the vector of reference temperatures for the current set of elements. Again, the order of these reference temperatures are determined by the sequence listed above and is assumed to hold in other modules. When all the element dependent drivers have been called by the **NLEMG** module driver, clean up operations begin. The entities that have been open for writing by the element routines are closed, the remaining in-core **DDVCT** and **DVCTD** tuples are written to the data base and the relations are sorted. If there are no design variables (all **DVID**'s are zero), the **DVCT** is sorted only on **KSIL**. Finally, if stress or strain constraints were defined in the bulk data stream, the **NLSMAT** matrix of constraint sensitivities to the displacements is closed. **NLSMAT** was opened by the **PREMAT** module when the **SCON** constraint flag was set.

Design Requirements:

1. The **MAKEST** module must have been called prior to the **NLEMG** module.

Error Conditions:

1. Illegal element geometries and nonexistent material properties are flagged.

Engineering Application Module: **NLLODGEN**

Entry Point: **NLLDGN**

Purpose:

To assemble the nonlinear design variable simple load vectors and nonlinear simple load sensitivities for all applied loads in the Bulk Data packet.

MAPOL Calling Sequence:

```
CALL NLLODGEN ( GSIZEB, GLBDES, DVCTD, DDVCT, DVSIZED, DDVSIZE,GMMCTD,
                DGMMCT, DMVID, DDMVI, TELMD, DTELM, TREFD, [DPTHVD],[DDPTHV],
                [DPGRVD], [DDPGRV] );
```

GSIZEB	The size of the structural set (Integer, Input)
GLBDES	Relation of global design variables (Character, Input)
DCVTD	Relation containing the data required for the assembly of the nonlinearly designed stiffness and mass matrices (Character, Input)
DDVCT	Relation containing the data required for the assembly of the nonlinearly designed sensitivity matrices (Character, Input)
DVSIZED	Unstructured entity containing memory allocation information on the **DVCTD** relation. (Character, Input)
DDVSIZE	Unstructured entity containing memory allocation information on the **DDVCT** relation. (Character, Input)
GMMCTD	Relation containing connectivity data for the **DMVID** design mass matrix (Character, Input)
DGMMCT	Relation containing connectivity data for the **DDMVI** mass sensitivity matrix (Character, Input)
DMVID	Unstructured entity containing the nonlinear mass matrix in a highly compressed format (Character, Input)
DDMVI	Unstructured entity containing the nonlinear mass sensitivity matrix in a highly compressed format (Character, Input)
TELMD	Unstructured entity containing the nonlinear design thermal load partitions (Character, Input)
DTELM	Unstructured entity containing the element nonlinear thermal load sensitivity matrix partitions (Character, Input)
TREFD	Unstructured entity containing the element reference temperatures for nonlinearly designed thermal loads. (Character, Input)
[DPTHVD]	Matrix entity containing the nonlinearly designed thermal loads (Character, Output)
[DDPTHV]	Matrix entity containing the nonlinear thermal load sensitivities (Character, Output)
[DPGRVD]	Matrix entity containing the nonlinearly designed gravity loads (Character, Output)

[DDPGRV]	Matrix entity containing the nonlinear gravity load sensitivities (Character, Output)

Application Calling Sequence:

None

Method:

Design Requirements:

Error Conditions:

Engineering Application Module: **NREDUCE**

Entry Point: **NREDUC**

Purpose:

To reduce the symmetric n-set stiffness, mass or loads matrix to the f-set if there are single point constraints in the boundary condition.

MAPOL Calling Sequence:

```
CALL NREDUCE ( [KNN], [PN], [PNSF(BC)], [YS(BC)], [KFF], [KFS],
               [KSS], [PF], [PS] );
```

[KNN]	Optional matrix containing the independent stiffness or mass matrix to be reduced (Input)
[PN]	Optional matrix containing the applied loads to be reduced (Input)
[PNSF(BC)]	The partitioning vector splitting the independent degrees of freedom into the free and the single point constraint degrees of freedom (Input), where **BC** represents the MAPOL boundary condition loop index number
[YS(BC)]	Optional matrix containing the vector of enforced displacements (Input), where **BC** represents the MAPOL boundary condition loop index number
[KFF]	Optional matrix containing the reduced form of **KNN** (Output)
[KFS]	Optional matrix containing the off-diagonal partition of **KFF** (Output)
[KSS]	Optional matrix containing the dependent diagonal partition of **KFF** (Output)
[PF]	Optional matrix containing the reduced form of **PN** (Output)
[PS]	The load matrix partition for computation of spcforces (Output)

Application Calling Sequence:

None

Method:

If the **PN** argument is nonblank, the module determines the number of columns in the loads matrix. Further, if the **YS** vector is nonblank, it is expanded to have the proper number of duplicate columns. Having taken care of the **YS** matrix, the module proceeds to check if the **KNN** argument is nonblank. If so, and there are no enforced displacements, the **KNN** matrix is partitioned into **KFF** and **KFS** (if the **KFS** matrix is input). If there are enforced displacements, the **KSS** partition is also saved if the **KSS** argument is supplied. The module then proceeds to reduce the loads matrix if the **PN** argument is nonblank. If there are no enforced displacements, the matrix is simply partitioned to **PF**. When enforced displacements are present, the loads on the free degrees of freedom are computed as:

$$[PF] = [\overline{PF}] - [KFS][YS]$$

The module then terminates.

Design Requirements:

1. If there are nonzero enforced displacements, the stiffness and loads reductions must be done concurrently or the **KFS** partition must be included in the loads call as input.

2. The **KFS** argument is always required when **YS** is nonblank.

Error Conditions:

None

Engineering Application Module: **OFPAEROM**

Entry Point: **OFPARO**

Purpose:

This module solves for the static aero applied loads on the aero boxes and for the displacements on the aero boxes to satisfy the **AIRDISP** and **TPRESSURE** print/punch requests. It loads the **OAGRDLOD** and **OAGRDDSP** relation.

MAPOL Calling Sequence:

```
CALL OFPAEROM ( NITER, BCID, MINDEX, SUB, GSIZE, GEOMSA, [GTKG], [GSTKG], QDP,
                [AIRFRC(MINDEX)], [DELTA(SUB)], [AICMAT(MINDEX)], [UAG(BC)],
                OAGRDLOD, OAGRDDSP );
```

NITER	Design iteration number (Optional, Integer, Input)
BCID	User defined boundary condition identification number (Integer, Input)
MINDEX	Mach number index associated with the current subscript (Integer, Input)
SUB	Current Mach number subscript number (Integer, Input)
GSIZE	Number of g-set DOF's including any that may have been added by **GDR** (Integer, Input)
GEOMSA	A relation describing the aerodynamic boxes for the steady aerodynamics **MODEL**. The location of the box centroid, normal and pitch moment axis are given. It is used in splining the aerodynamics to the structure and to map responses back to the aerodynamic boxes (Input)
[GTKG]	The matrix of splining coefficients relating the aerodynamic pressures to forces at the structural grids (Input)
[GSTKG]	The matrix of splining coefficients relating the structural displacements to the streamwise slopes of the aerodynamic boxes (Input)
QDP	Dynamic pressure associated with the current subscript (Real, Input)
[AIRFRC(MINDEX)]	Matrix containing the aerodynamic forces for unit configuration parameters for the current Mach number index. If both symmetric and antisymmetric conditions exist for the Mach number, both sets of configuration parameters will coexist in **AIRFRC** (Input)
[DELTA(SUB)]	Matrix containing the set of configuration parameters representing the user input fixed values and the trimmed unknown values for the **SUB** subscript's trim cases (Input)
[AICMAT(MINDEX)]	Matrix containing the steady aerodynamic influence coefficients for either symmetric or antisymmetric Mach numbers as appropriate for the symmetry of the cases in the current boundary condition (Input)
[UAG(BC)]	Matrix of static displacements for all **SAERO** subcases in the current boundary condition in the order the subcases appear in the **CASE** relation (Input), where **BC** represents the MAPOL boundary condition loop index number

OAGRDLOD	A relation containing the rigid, flexible correction and flexible forces and pressures for each **SAERO** subcase for the trimmed configuration parameters. Outputs are for the aerodynamic elements whose **TPRESSURE** output was requested in Solution Control. These constitute the loads of the "trimmed" state of the configuration. (Output)
OAGRDDSP	A relation containing the displacements for each **SAERO** subcase's set of configuration parameters for the aerodynamic elements whose **AIRDISP** output was requested in Solution Control. These constitute the trimmed displacements of the aerodynamic **MODEL**. (Output)

Application Calling Sequence:

None

Method:

The **CASE** relation is read to obtain the list of all **SAERO** subcases for the current boundary condition. The **AIRDISP** and **TPRESSURE** print/punch requests are checked and the module terminates if no output requests exist.

If output is needed, the **TRIM** relation is read to obtain the subscript values of each subcase. A partitioning vector is formed as the **TRIM** data are searched to extract the proper columns from the **UAG** matrix for the subcases associated with the current **SUB** value. Then, for each subcase to be processed, the particular print and punch requests are evaluated and, in the most general case, the following are computed:

Rigid Air Loads:

$$= \mathbf{QDP*[AIRFRC][DELTA]}$$

Flexible Correction to the Rigid Air Loads:

$$= \mathbf{QDP*[AICMAT]^T[GSTKG]^T[UAG]}$$

Total Applied Air Loads:

$$= \mathbf{Rigid + Flexible}$$

Displacements on the aero boxes

$$= \mathbf{[GTKG]^T[UAG]}$$

where in each case the **[DELTA]** and **[UAG]** matrices are partitioned to include only the relevant subcases for the current subscript.

Finally, the scratch matrices on which these results reside are read and output to the **OAGRDLOD** and **OAGRDDSP** relations for the loads and displacements, respectively.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **OFPALOAD**

Entry Point: **OFPALD**

Purpose:

Solves for the static aero applied loads and **SPC** forces to satisfy the print/punch requests. The resultant loads are written to the **OGRIDLOD** relation.

MAPOL Calling Sequence:

```
CALL OFPALOAD ( NITER, BCID, MINDEX, SUB, GSIZE, BGPDT(BC), [GTKG], [GSTKG],
                QDP, [AIRFRC(MINDEX)], [DELTA(SUB)], [AICMAT(MINDEX)],
                [UAG(BC)], [MGG], [AAG(BC)], [KFS], [KSS], [UAF], [YS(BC)],
                [PNSF(BC)], [PGMN(BC)], [PFJK], NGDR, USET(BC), OGRIDLOD );
```

NITER	Design iteration number (Integer, Input)
BCID	User defined boundary condition identification number (Integer, Input)
MINDEX	Mach number index for the current subscript value (Integer, Input)
SUB	Subscript number of **SAERO** subcases considered in this cal. (Integer, Input)
GSIZE	Number of degrees of freedom in the g-set including those that may have been added by **GDR** (Integer, Input)
BGPDT(BC)	Relation of basic grid point data for the boundary condition (including any extra points and **GDR** scalar points which may be added by the **GDR** module) (Input), where **BC** represents the MAPOL boundary condition loop index number
[GTKG]	The matrix of splining coefficients relating the aerodynamic pressures to forces at the structural grids (Input)
[GSTKG]	The matrix of splining coefficients relating the structural displacements to the streamwise slopes of the aerodynamic boxes (Input)
QDP	Dynamic pressure associated with the current subscript (Real, Input)
[AIRFRC(MINDEX)]	Matrix containing the aerodynamic forces for unit configuration parameters for the current Mach number index. If both symmetric and antisymmetric conditions exist for the Mach number, both sets of configuration parameters will coexist in **AIRFRC** (Input)
[DELTA(SUB)]	Matrix containing the set of configuration parameters representing the user input fixed values and the trimmed unknown values for the **SUB** subscript's trim cases (Input)
[AICMAT(MINDEX)]	Matrix containing the steady aerodynamic influence coefficients for either symmetric or antisymmetric Mach numbers as appropriate for the symmetry of the cases in the current boundary condition (Input)
[UAG(BC)]	Matrix of static displacements for all **SAERO** subcases in the current boundary condition in the order the subcases appear in the **CASE** relation (Input), where **BC** represents the MAPOL boundary condition loop index number
[MGG]	Mass matrix in the g-set (Input)

[AAG(BC)]	Matrix of accelerations for all **SAERO** subcases in the current boundary condition in the order the subcases appear in the **CASE** relation (Input), where **BC** represents the MAPOL boundary condition loop index number
[KFS]	The off-diagonal matrix partition of the independent degrees of freedom that results from the **SPC** partitioning (Input)
[KSS]	The dependent DOF diagonal matrix partition of the independent degrees of freedom that results from the **SPC** partitioning (Input)
[UAF]	Matrix of free (f-set) static displacements for all **SAERO** subcases in the current boundary condition in the order the subcases appear in the **CASE** relation (Input)
[YS(BC)]	Vector of enforced displacements for the boundary condition (one column) (Input)
[PNSF(BC)]	Partitioning vector to divide the independent DOFs into the free and **SPC** DOFs (Input), where **BC** represents the MAPOL boundary condition loop index number
[PGMN(BC)]	Partitioning vector to divide the g-set DOFs into the **MPC** and independent DOF's (Input), where **BC** represents the MAPOL boundary condition loop index number
[PFJK]	Partitioning vector to divide the f-set DOFs that may include **GDR** generated scalar points into the original f-set DOF's
NGDR	Denotes dynamic reduction in the boundary condition. (Input, Integer) = **0** No **GDR** = **-1** **GDR** is used
USET(BC)	The unstructured entity of DOF masks for all the points in the current boundary conditions (Input), where **BC** represents the MAPOL boundary condition loop index number
OGRIDLOD	Relation of loads on structural grid points (Output)

Application Calling Sequence:

None

Method:

First the **CASE** relation entries for **SAERO** subcases in the current boundary condition are read. Then the **TRIM** relation is read to determine which subcases are associated with the current subscript value. Then the output **LOAD** and **SPCF** print/punch requests are examined to see if any further work is needed. If no print or punch requests are needed for the subcases associated with the **SUB**'th subscript, control is returned to the MAPOL sequence.

If **SPCF** requests exist, the preliminary computations are performed in the **ARSPCF** module. It computes:

$$[QGV1] = [KFS]^T\{UF\} + [KSS]\{YS\}$$

for all the appropriate columns of UAF that are associated with the **SUB**'th subscript. The input **YS** vector is expanded to contain the correct number of columns.

Then the computation of the applied loads is done. First, the **BGPDT** data are read and the **OGRIDLOD** relation is opened for output. Then the loads for each subcase in the subscript is solved for subject to the existence of a print request for that subcase (either **LOAD** or **SPCF**). The following loads are computed:

Rigid Air Loads on the Structural Grids

$$= \mathbf{QDP^*[GTKG][AIRFRC][DELTA]}$$

Flexible Correction to the Rigid Air Loads

$$= \mathbf{QDP^*[GTKG][AIC]^T[GSTKG]^T[UAG]}$$

Total Applied Load

$$= \mathbf{Rigid + Flexible}$$

Inertial Load

$$= \mathbf{-[MGG][AAG]}$$

Where the appropriate inputs are not available, the computations are simply ignored with no warning. Thus, the optional calling arguments may be used to perform parts of the computations without requiring that all pieces be provided.

Then, the output **LOAD**s matrices are opened and the **CASE LOAD**s print and punch requests are used to load the **OGRIDLOD** relation with the **RIGID**, **FLEXIBLE**, **APPLIED** and **INERTIA** loads.

Finally, if any **SPCF** output requests exist the **APPLIED** loads that were computed are combined with the **QGV1** terms to result in the **SPC** reaction forces:

$$\mathbf{[SPCF] = [QGV1] - [Applied\ load]}$$

For each DOF for which **SPC** forces have been requested.

Design Requirements:

1. **SPC** force computations for other disciplines occur in the **OFPSPCF** module.

2. Only those arguments that are present will be used. If data are missing, the dependent terms will be omitted from the output.

Error Conditions:

None

Engineering Application Module: **OFPDISP**

Entry Point: **OFPDSP**

Purpose:

To print selected displacements, velocities and/or accelerations from any analyses in the current boundary condition.

MAPOL Calling Sequence:

```
CALL OFPDISP ( BCID, NITER, GSIZE, BGPDT(BC), ESIZE(BC), PSIZE(BC),
               OGRIDDSP, [UG(BC)], [AG(BC)], [UAG(BC)], [AAG(BC)], [BLUG],
               [BLUE], [UTRANG], [UTRANE], [UFREQG], [UFREQE], LAMBDA,
               [PHIG(BC)], [PHIBG(BC)], LSTFLG );
```

BCID	User defined boundary condition identification number (Integer, Input)
NITER	Iteration number for the current design iteration (Integer, Input)
GSIZE	The size of the structural set (Integer, Input)
BGPDT(BC)	Relation of basic grid point coordinate data (Input), where **BC** represents the MAPOL boundary condition loop index number
ESIZE(BC)	The number of extra point degrees of freedom in the boundary condition (Integer, Input), where **BC** represents the MAPOL boundary condition loop index number
PSIZE(BC)	The size of the physical set for the current boundary condition. (Integer, Input), where **BC** represents the MAPOL boundary condition loop index number
OGRIDDSP	Relation for storage of displacement data (Input)
[UG(BC)]	Matrix of global displacements from **STATICS** analyses (Input), where **BC** represents the MAPOL boundary condition loop index number
[AG(BC)]	Matrix of global accelerations from **STATICS** analyses (Input), where **BC** represents the MAPOL boundary condition loop index number
[UAG(BC)]	Matrix of global displacements from **SAERO** analyses (Input), where **BC** represents the MAPOL boundary condition loop index number
[AAG(BC)]	Matrix of global accelerations from **SAERO** analyses (Input), where **BC** represents the MAPOL boundary condition loop index number
[BLUG]	Matrix of global displacements/velocities/accelerations for **BLAST** response analyses (Input)
[BLUE]	Matrix of extra point displacements/velocities/ accelerations for **BLAST** response analyses (Input)
[UTRANG]	Matrix of global displacements/velocities/ accelerations for **TRANSIENT** response analyses (Input)
[UTRANE]	Matrix of extra point displacements/velocities/ accelerations for **TRANSIENT** response analyses (Input)
[UFREQG]	Matrix of global displacements/velocities/ accelerations for **FREQUENCY** response analyses (Input)

[UFREQE]	Matrix of extra point displacements/velocities/ accelerations for **FREQUENCY** response analyses (Input)
LAMBDA	Relational entity containing the output from the real eigenanalysis (Input)
[PHIG(BC)]	Matrix of global eigenvectors from real eigenanalysis for **MODES** analyses (Input), where **BC** represents the MAPOL boundary condition loop index number
[PHIGB(BC)]	Matrix of global eigenvectors for **BUCKLING** analyses (Input), where **BC** represents the MAPOL boundary condition loop index number
LSTFLG	Integer flag to indicate if for last iteration output only (Integer, Input) = **1** for last iteration only = **0** other general cases

Application Calling Sequence:

None

Method:

The module begins by reading the **CASE** relation nodal response quantity print options for the current boundary condition. The following print requests are treated in the **OFPDISP** module:

(1) **DISPLACEMENT**

(2) **VELOCITY**

(3) **ACCELERATION**

(4) **ROOTS** (for normal modes analyses)

As the **CASE** data are searched, the **FLTFLG** and **MODFLG** logicals are set to **TRUE** if either **FLUTTER** or **MODES** disciplines are associated with these print requests. If no prints are requested, the module terminates, otherwise, the **ITERLIST**, **GRIDLIST**, **MODELIST**, **TIMELIST** and **FREQLIST** data are prepared for easy retrieval in determining which nodes and subcases are requested in each case.

The **BGPDT** data are then read into open core and the number of extra point degrees of freedom in the current boundary condition is determined. Finally, the code checks to see if any flutter displacements (eigenvectors) have been requested for an optimization boundary condition. If so, the request is explicitly turned off since ASTROS does not compute the eigenvector for optimization boundary conditions. The next segment of code is set aside for special discipline dependent processing. In this module, the flutter eigenvector print requires the transformation of the modal participation factors for any flutter eigenvectors into physical coordinates using the input **PHIG** matrix and the **FLUTREL** and **FLUTMODE** entities that were created in the **FLUTTRAN** module. Again, if no flutter conditions were found in the analysis, the module explicitly turns off the print request. Otherwise, the physical mode shape is computed and stored in a pair of scratch entities: one for the structural degrees of freedom and one for the extra point degrees of freedom.

The main loop in the module now begins. This loop is over all the disciplines that have nodal response quantities. For each discipline, there is a loop over all the **CASE** tuples retrieved at the beginning of the module. Only those **CASE** tuples matching the current discipline are treated at each pass of the outermost loop. The **DSPSUB** submodule is called for each **CASE** tuple to determine the number and identification numbers for each subcase for which output is desired. A subcase is considered to be one displacement/velocity/acceleration vector for a particular time step, frequency step, load condition, etc. Then, depending on the nature of the discipline, one of five print routines is called to read into memory the proper nodal vector and print the terms to the user output file. Once all the subcases for the current **CASE** tuple have

been processed, the **CASE** tuple loop continues for the current discipline. When all disciplines or all **CASE** tuples have been processed, the module terminates.

Design Requirements:

1. The **OFPDISP** module is designed to be called at the conclusion of the boundary condition loop when all the physical nodal response quantities have been computed for all the analyzed disciplines.

Error Conditions:

None

Engineering Application Module: **OFPDLOAD**

Entry Point: **OFPDLD**

Purpose:

Processes the solution control load output requests for the current boundary condition for dynamic loads (transient, frequency and gust) and stores the loads on the physical degrees of freedom to the **OGRIDLOD** relation for those subcases and grids selected in solution control.

MAPOL Calling Sequence:

```
CALL OFPDLOAD ( NITER, BCID, BGPDT(BC), PSIZE(BC), ESIZE(BC), [PHIG(BC)],
                [PTGLOAD], [PTHLOAD], [PFGLOAD], [PFHLOAD], OGRIDLOD );
```

NITER	Current design iteration number (Integer, Input)
BCID	User defined boundary condition identification number (Integer, Input)
BGPDT(BC)	Relation of basic grid point data for the boundary condition (including any extra points and **GDR** scalar points which may be added by the **GDR** module) (Input), where **BC** represents the MAPOL boundary condition loop index number
PSIZE(BC)	The size of the physical set for the current boundary condition (Integer, Input), where **BC** represents the MAPOL boundary condition loop index number
ESIZE(BC)	Number of extra point DOF's defined for the boundary condition (Integer, Output), where **BC** represents the MAPOL boundary condition loop index number
[PHIG(BC)]	Matrix of normal mode eigenvectors in the structural g-set (Input), where **BC** represents the MAPOL boundary condition loop index number
[PTGLOAD]	Matrix of g-set applied dynamic loads for the direct transient analyses in the current boundary condition (Input)
[PTHLOAD]	Matrix of h-set applied dynamic loads for the modal transient **GUST** analyses in the current boundary condition (Input)
[PFGLOAD]	Matrix of g-set applied dynamic loads for the direct frequency analyses in the current boundary condition (Input)
[PFHLOAD]	Matrix of h-set applied dynamic loads for the modal frequency analyses with **GUST** in the current boundary condition (Input)
OGRIDLOD	Relation of applied loads on structural grid points (Output)

Application Calling Sequence:

None

Method:

The **CASE** relation is read for all transient and frequency response analysis and the **LOADPRNT** print and punch requests for **LOAD**s are examined. If any requests exist, processing continues by opening the **BGPDT** and reading the internal/external point identifications to allow storing the matrix data or the **OGRIDLOD** relation labelled with the external ids.

If any **GUST** loads are requested, the modal dynamic loads are transformed to the physical degrees of freedom as:

[PGUSTT] = [PHIG] [PTHLOAD] for transient gust
[PGUSTF] = [PHIG] [PFHLOAD] for harmonic gust

To perform these operations, the normal modes must be expanded to include extra points for the single subcase of transient and or frequency that is allowed. Then the multiplications are performed.

Finally, once all the direct matrices are available, the **CASE** control print requests are processed, the corresponding columns are identified by interpreting the **TIME** or **FREQ** options and the **GRIDLIST** data are read to determine which points are chosen. The terms are then written to the **OGRIDLOD** relation as **APPLIED** loads.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **OFPEDR**

Entry Point: **OFPEDR**

Purpose:

To print selected element stress, strain, force and/or strain energies from any analyses in the current boundary condition.

MAPOL Calling Sequence:

```
CALL OFPEDR ( BCID, HSIZE(BC), NITER, LSTFLG );
```

BCID	User defined boundary condition identification number (Integer, Input)
HSIZE(BC)	Number of modal dynamic degrees of freedom in the current boundary condition (Input), where **BC** represents the MAPOL boundary condition loop index number
NITER	Iteration number for the current design iteration (Integer, Input)
LSTFLG	Integer flag to indicate if for last iteration output only (Integer, Input) = **1** for last iteration only = **0** other general cases

Application Calling Sequence:

None

Method:

The module begins by reading the **CASE** relation element response quantity print options for the current boundary condition. The following print requests are treated in the **OFPEDR** module:

(1) **STRESS**

(2) **STRAIN**

(3) **FORCE**

(4) **ENERGY**

If no prints are requested, the module terminates, otherwise, the **ITERLIST**, **ELEMLIST**, **MODELIST**, **TIMELIST** and **FREQLIST** data are prepared for easy retrieval in determining which elements and subcases are requested in each case. The main loop in the module now begins. This loop is over all the disciplines that have element response quantities.

For each discipline, there is a loop over all the **CASE** tuples retrieved at the beginning of the module. Only those **CASE** tuples matching the current discipline are treated at each pass of the outermost loop. The **OFPSUB** submodule is called for each **CASE** tuple to determine the number and identification numbers for each subcase for which output is desired. A subcase is considered to be one displacement vector for a particular time step, frequency step, load condition, etc. For each subcase, the set of element response print utilities (one for each element type) are called for each of the four quantities that can be printed. If the strain energy is requested, the **OFPESE** submodule is called to compute the total strain energy for the current displacement field as a preface operation prior to the element dependent print routines. Once all the quantities for all the subcases for the current **CASE** tuple have been processed, the **CASE** tuple loop continues for the current discipline. When all disciplines or all **CASE** tuples have been processed, the module terminates.

Design Requirements:

1. The **OFPEDR** module is designed to be called at the conclusion of the boundary condition loop when all the physical nodal response quantities have been computed for all the analyzed disciplines.

2. The **EDR** module must have been called to store the computed element response quantities onto the **EOxxxxx** entities which are read by the **OFPEDR** module.

Error Conditions:

None

Engineering Application Module: **OFPGRAD**

Entry Point: **OFPGRA**

Purpose:

Stores the data necessary to satisfy the solution control print and punch requests **OGRADIENT** and **CGRADIENT** (objective function gradient and constraint gradient, respectively).

MAPOL Calling Sequence:

```
CALL OFPGRAD ( NITER, [AMAT], GLBDES, CONST, CONSTORD, GRADIENT );
```

NITER	Design iteration number (Integer, Input)
[AMAT]	The matrix of constraint gradients for active constraints in the current design iteration (Input)
GLBDES	The relation of global design variable values and objective function sensitivities for all design iterations that have been analyzed. (Input)
CONST	The relation of applied design constraints for all design iterations (Input)
CONSTORD	The relation of reordered design constraints for the current design iteration (Input)
GRADIENT	The relation of output constraint gradients for the requested constraints and design variables that satisfy the Solution Control **CGRADIENT** and **OGRADIENT** output requests (Output)

Application Calling Sequence:

None

Method:

The **OPTIMIZE** relation is read to determine if any **OGRADIENT** or **CGRADIENT** print or punch requests exist. If they do, processing continues by determining if this iteration is in the set of iterations selected. If it is, the the **AMAT** matrix is opened and read into memory as are the **GLBDES** entries for the current iteration. The **CONST** relation is read into memory and reordered to match the **AMAT** matrix. Then the **GRADIENT** entity is loaded with the objective or constraint gradient terms for the requested constraints and global design variables.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **OFPLOAD**

Entry Point: **OFPLOD**

Purpose:

To print selected applied external loads from any analyses in the current boundary condition.

MAPOL Calling Sequence:

```
CALL OFPLOAD ( BCID, NITER, GSIZE, BGPDT(BC), PSIZE(BC), [PG],
               TRMTYP, QDP, [GTKG], [AIRFRC(MINDEX)], [DELTA] );
```

BCID	User defined boundary condition identification number (Integer, Input)
NITER	Iteration number for the current design iteration (Integer, Input)
GSIZE	The size of the structural set (Integer, Input)
BGPDT(BC)	Relation of basic grid point coordinate data (Input), where **BC** represents the MAPOL boundary condition loop index number
PSIZE(BC)	The size of the physical set for the current boundary condition. (Integer, Input), where **BC** represents the MAPOL boundary condition loop index number
[PG]	Matrix of applied loads for **STATICS** analyses in the current boundary condition (Input)
TRMTYP	The trim type for the steady aeroelastic analyses = **0** zero degree of freedom trim = **1** lift only trim = **2** lift/pitching moment trim (Integer, Input)
QDP	Dynamic pressure for the **SAERO** analyses in the current boundary condition (Real, Input)
[GTKG]	Matrix containing the steady aerodynamic spline in the structural set (Input)
[AIRFRC(MINDEX)]	Matrix containing the aerodynamic forces for unit configuration parameters for the current Mach number and Symmetry (Input)
[DELTA]	Matrix containing the configuration parameter values resulting from the current trim condition (Input)

Application Calling Sequence:

None

Method:

The module begins by reading the **CASE** relation applied load print options for the current boundary condition. The **LOAD** print requests are treated in the **OFPLOAD** module for all ASTROS disciplines. As the **CASE** data are searched, the **AROFLG** logical is set to **TRUE** if any **SAERO** cases with a **TRMTYP** greater than zero are found. If no prints are requested, the module terminates, otherwise, the **GRIDLIST**, **MODELIST**, **TIMELIST** and **FREQLIST** data are prepared for easy retrieval in determining which nodes and subcases are requested in each case. The **BGPDT** data are then read into open core and the number of extra point degrees of freedom in the current boundary condition is determined. The next segment of code is set aside for special discipline dependent processing. In this module, the steady air loads associated with **TRIM** analyses must be computed from the **AIRFRC** matrix of loads due to "unit" configuration parameters and the **DELTA** matrix of trimmed configuration parameters. The result must

then be splined to the structural degrees of freedom using the **GTKG** spline transformation matrix. The structural applied loads are stored in a scratch entity for use in the subsequent print processing. The main loop in the module now begins. This loop is over all the disciplines that have applied loads. For each discipline, there is a loop over all the **CASE** tuples retrieved at the beginning of the module. Only those **CASE** tuples matching the current discipline are treated at each pass of the outermost loop. The **LODSUB** submodule is called for each **CASE** tuple to determine the number and identification numbers for each subcase for which output is desired. A subcase is considered to be one load vector for a particular time step, frequency step, load condition, etc. Then, depending on the nature of the discipline, one of two print routines is called to read into memory the proper vector and to print the terms to the user output file. Once all the subcases for the current **CASE** tuple have been processed, the **CASE** tuple loop continues for the current discipline. When all disciplines or all **CASE** tuples have been processed, the module terminates.

Design Requirements:

1. The **OFPLOAD** module is designed to be called at the conclusion of the boundary condition loop.

Error Conditions:

None

Engineering Application Module: **OFPMROOT**

Entry Point: **OFPMRT**

Purpose:

Processes the solution control normal modes root output requests.

MAPOL Calling Sequence:

```
CALL OFPMROOT ( NITER, BCID, LAMBDA, LASTFLAG );
```

NITER	Current design iteration number. (Integer, Input)
BCID	User defined boundary condition identification number (Integer, Input)
LAMBDA	The relation of normal modes eigenvalues for all boundary conditions and design iterations (Input)
LASTFLAG	An optional argument which, if nonzero, implies that the call is being made only to satisfy **ITER=LAST** print or punch requests (Integer, Input)

Application Calling Sequence:

None

Method:

The **CASE** relation is read to obtain the print/punch requests for **ROOTS**. If any requests exist, they are processed. If the **LASTFLAG** is nonzero, only those requests in which the **ITER=LAST** flag is set in the **ROOTPRNT CASE** relation attribute are considered.

For the modes selected by the **MODELIST**, the **OEIGS** and **LAMBDA** entities are read and the eigenvalue extraction summary table and the extracted eigenvalues are printed to the output file. Punch requests are ignored since the data are stored already on the **LAMBDA** relation.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **OFPSPCF**

Entry Point: **OFPSPF**

Purpose:

Recovers single-point forces of constraint and loads the results to the **OGRIDLOD** relation

MAPOL Calling Sequence:

```
CALL OFPSPCF ( NITER, BCID, BC, DISC, CMPLX, GSIZE, ESIZE(BC), NGDR, [KFS],
               [KSS], [UF], [YS(BC)], [PS], [PNSF(BC)], [PGMN(BC)], [PFJK],
               [PHIG(BC)], [PGLOAD], [PHLOAD], BGPDT(BC), OGRIDLOD );
```

NITER	Current design iteration number (Optional, Integer, Input)
BCID	User defined boundary condition identification number (Integer, Input)
BC	The MAPOL boundary condition loop index number (Integer, Input)
DISC	Integer key indicating the discipline whose **SPC** forces are to be recovered. **=1** for statics **=2** for modes **=4** for flutter **=5** for transient analysis **=6** for frequency analysis **=8** for nuclear blast Note that static aeroelasticity (**DISC=3**) is supported in the **OFPALOAD** module. (Integer, Input)
CMPLX	Integer flag indicating whether the discipline's displacement field is real (**=1**) or complex (**=2**) (Integer, Input)
GSIZE	Number of degrees of freedom in the g-set including those that may have been added by **GDR** (Integer, Input)
ESIZE(BC)	Number of extra point DOF's defined for the boundary condition (Integer, Output)
NGDR	Denotes dynamic reduction in the boundary condition. (Input, Integer) **= 0** No **GDR** **= -1 GDR** is used
[KFS]	The off-diagonal matrix partition of the independent degrees of freedom that results from the **SPC** partitioning (Input)
[KSS]	The dependent DOF diagonal matrix partition of the independent degrees of freedom that results from the **SPC** partitioning (Input)
[UF]	Matrix of free (f-set) static displacements for all the **DISC** subcases in the current boundary condition in the order the subcases appear in the **CASE** relation (Input)
[YS(BC)]	Vector of enforced displacements for the boundary condition (one column) (Optional, Input)
[PS]	Matrix of static loads applied to the **SPC** DOF's (Partition of the free DOF loads matrix) (Optional, Input)
[PNSF(BC)]	Partitioning vector to divide the independent DOFs into the free and **SPC** DOFs (Input)

[PGMN(BC)]	Partitioning vector to divide the g-set DOFs into the **MPC** and independent DOF's (Input)
[PFJK]	Partitioning vector to divide the f-set DOFs that may include **GDR** generated scalar points into the original f-set DOF's (Optional, but required if **NGDR <>0**; Input)
[PHIG(BC)]	Matrix of normal mode eigenvectors in the structural g-set (Optional, Input)
[PGLOAD]	Matrix of g-set applied dynamic loads for the direct transient or frequency analyses (as appropriate for **DISC**) in the current boundary condition (Optional, Input)
[PHLOAD]	Matrix of h-set applied dynamic loads for the modal transient or frequency **GUST** analyses (as appropriate for **DISC**) in the current boundary condition (Optional, Input)
BGPDT(BC)	Relation of basic grid point data for the boundary condition (including any extra points and **GDR** scalar points which may be added by the **GDR** module) (Input), where **BC** represents the MAPOL boundary condition loop index number
OGRIDLOD	Relation of loads on structural grid points (Output)

Application Calling Sequence:

None

Method:

This module computes the **SPC** reaction forces for all disciplines in ASTROS except **SAERO** and **NPSAERO**. **NPSAERO** has no structural loads and the **SAERO SPC** forces are computed in the **OFPALOAD** module where the applied loads (an input to the **SPC** computations) are computed.

First the **CASE** relation is read for all entries with a **DISFLAG** of **DISC** for the current boundary condition. Then the **SPCF** print requests are examined to determine if any output is needed for this discipline, design iteration, etc. If not, the module terminates otherwise computations continue with the creation of scratch entities to hold the constituent parts of the **SPC** calculations. The **BGPDT** data are read into memory and the **OGRIDLOD** relation is opened in preparation for output.

The existence of enforced displacements, **YS** and loads on the **SPC** dofs, **PS** is checked and logical flags are set for downstream computations. If **GDR** was used (as indicated by **NGDR <>0**), the **PFJK** partition matrix is used to extract the original f-set DOF from **UF** from the input set which includes **GDR** scalar points.

Then some discipline dependent processing takes place. If **DISC = 4** (**FLUTTER**), the **FLMODE** hidden entity is read and the flutter eigenvectors (if any) are read, stripped of the extra point degrees of freedom and reduced to the f-set. Transient and frequency disciplines require special processing because of the nature of the displacement matrices (containing velocities and accelerations). This processing is done in **DYSPCF** and results in a g-set sized matrix of the loads applied to the **SPC** DOFs for each time or frequency step. **GUST** loads are treated here to recover the direct applied loads from the **PHLOAD** input. Extra points are partitioned out of these loads matrices if needed.

Then the actual recovery process begins. First the **QSV** matrix of **SPC** forces are computed from the appropriate constituent terms

$$[QSV] = [KFS]^T\{UF\} + [KSS]\{YS\} - \{PS\}$$

where **YS** has been expanded to have the appropriate number of columns and the proper terms are ignored if **YS** or **PS** is blank or empty.

Then the **QSV** matrix is expanded to the g-set, the nonzero terms are read and compared to the output requests and the appropriate terms are loaded to the **OGRIDLOD** relation. For the dynamic response disciplines, the applied loads **PS** are extracted from the g-set output of the **DYSPCF** submodule and the reaction forces are adjusted accordingly.

Design Requirements:

1. **SAERO** single point constraint reactions are computed in the **OFPALOAD** module where the applied loads are computed.

Error Conditions:

None

Engineering Application Module: **PBKLEVAL**

Entry Point: **BKLEVA**

Purpose:

Evaluates the current values of the panel buckling constraints.

MAPOL Calling Sequence:

```
CALL PBKLEVAL ( BCID, NITER, NDV, GLBDES, LOCLVAR, [PTRANS], CONST,
                PDLIST, OPNLBUCK );
```

BCID	User defined boundary condition identification number (Integer, Input)
NITER	Design iteration number (Integer,Input)
NDV	Number of design variables (Integer,Input)
GLBDES	Relation of global design variables (Character,Input)
LOCLVAR	Relation containing the relationship between local variables and global variables in the design problem (Character,Input)
[PTRANS]	The design variable linking matrix (Character,Input)
CONST	Relation of constraint values (Character,Output)
PDLIST	Relation containing panel buckling constraint sensitivity information.(Character,Output)
OPNLBUCK	Relation containing panel buckling constraint output (Character,Output)

Application Calling Sequence:

None

Method:

This module first checks if any **DCONBK** bulk data entries referenced by any STATICS and/or SAERO disciplines at the current boundary condition to determine if there are any panel buckling constraints applied. If any are found, the QUAD4 and/or TRIA3 element data are obtained from relation **QUAD4EST** and/or **TRIA3EST**; the element force data are obtained from relation **EOQD4** and/or **EOTR3**; and the panel buckling constraint values are evaluated and stored into relation **CONST**. The constraint sensitivity data are also prepared in this module.

Design Requirements:

Because this module requires the element output relation from module **EDR**, it should only be called after that module.

Error Conditions:

The element has no force data for buckling constraint evaluation is flagged.

Engineering Application Module: **PBKLSENS**

Entry Point: **BKSENS**

Purpose:

Evaluates the panel buckling constraint sensitivity.

MAPOL Calling Sequence:

```
CALL PBKLSENS ( BCID, NITER, NDV, GLBDES, LOCLVAR, [PTRANS], PDLIST, [AMAT] );
```

BCID	User defined boundary condition identification number (Integer, Input)
NITER	Design iteration number (Integer,Input)
NDV	Number of design variables (Integer,Input)
GLBDES	Relation of global design variables (Character,Input)
LOCLVAR	Relation containing the relationship between local variables and global variables in the design problem (Character,Input)
[PTRANS]	The design variable linking matrix (Character,Input)
PDLIST	Relation containing panel buckling constraint sensitivity information (Character,Input)
[AMAT]	Matrix containing the sensitivity of the constraints to changes in the design variable (Character,Output)

Application Calling Sequence:

None

Method:

This module first check if any active panel buckling constraints for the current boundary condition and obtained the prepared constraint sensitivity data from relations **PDLIST** and **CONST**. Then the local design variable data is obtained from relation **LOCLVAR**. For each active panel buckling constraint, the sensitivity to the design variables is computed and stored into matrix **[AMAT]**.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **PFBULK**

Entry Point: **PFBULK**

Purpose:

To perform a number of preface operations to form additional collections of data and to make error checks not done in **IFP** to identify input errors before costly analyses are performed.

MAPOL Calling Sequence:

```
CALL PFBULK ( GSIZEB, EOSUMMRY, EODISC, GPFELEM );
```

GSIZEB	Length of the g-set vectors (Integer, Input)
EOSUMMRY	Relational entity containing the summary of entities for which element response quantities are desired (Output)
EODISC	Unstructured entity referred to by an attribute of **EOSUMMRY** containing the set of disciplines and subcases for the element response quantities
GPFELEM	Relational entity containing the set of elements connected to grid points for which grid point forces are desired (Output)

Application Calling Sequence:

None

Method:

This module first computes the coordinate functions and centroid functions which are required by any user function constraints. Those response values are stored into relational entities to be used by the function evaluation utilities. Then the module performs tests on selected bulk data entities to see if they contain data. If they do, the indicated subroutine is called to generate further data and perform error checks:

BULK DATA	SUBROUTINE	GENERATED ENTITY
TEMP, TEMPD	PRETMP	GRIDTEMP
FREQ, FREQ1, FREQ2	PREFRQ	FREQL
TSTEP	PRETST	OTL

The module also checks that constraint requests specified in the **FLUTTER** solution control command have corresponding **DCONFLT** bulk data entries.

As a final step, the **PFBULK** module performs the preliminary processing of solution control print requests that depend on elements. These include all the element response quantities (i.e., stress or strain) and the grid point force balance. The first stage is performed in the **PREGPF** submodule which builds the **GPFELEM** relation from the element connectivity data and the sets of nodes for which a force balance in requested. Then the **PREEDR** submodule is called to build the **EOSUMMRY** and **EODISC** entities which list those elements for which element data recovery should be performed in the **EDR** module. These entities are also used in **OFPEDR** to direct the printing of the computed quantities.

Design Requirements:

1. This is a preface module that called after **EMG** and **MAKEST**

Error Conditions:

None

Engineering Application Module: **QHHLGEN**

Entry Point: **QHHGEN**

Purpose:

To compute the discipline dependent unsteady aerodynamic matrices for gust analyses in the modal dynamic degrees of freedom.

MAPOL Calling Sequence:

```
CALL QHHLGEN ( BCID, ESIZE(BC), [QKKL], [QKJL], [UGTKA], [PHIA], [PHIKH],
               [QHHL], [QHJL] );
```

BCID	User defined boundary condition identification number (Integer, Input)
ESIZE(BC)	The number of extra point degrees of freedom in the boundary condition (Integer, Input), where **BC** represents the MAPOL boundary condition loop index number
[QKKL]	Matrix list containing the matrix product (Output): $[SKJ][AJJT]^{-T}([D1JK] + ik[D2JK])$ used for flutter and gust analyses (Input)
[QKJL]	Matrix list containing the matrix product: $[SKJ][AJJT]^{-T}$ used for gust analyses (Input)
[UGTKA]	Matrix containing the unsteady aerodynamic spline in the analysis set (Input)
[PHIA]	Matrix containing the real eigenvectors in the analysis set (Input)
[PHIKH]	Matrix containing the matrix product (Output): $[UGTKA][PHIA]$ with the analysis set expanded to include extra points (Output)
[QHHL]	The modal unsteady aerodynamic influence coefficients for gust (Output): $[PHIKH]^{T}[QKK][PHIKH]$
[QHJL]	The modal unsteady aerodynamic influence coefficients for gust (Output): $[PHIKH]^{T}[QKJ]$

Application Calling Sequence:

None

Method:

The **QHLLGEN** module begins by retrieving all the **CASE** tuples for the current boundary condition. The number of gust options on transient or frequency response disciplines are counted to determine what actions are required by the module. If gust conditions do not exist, control returns to the executive. If **QZHH** and **QJH** are required, the module continues by reading the **BGPDT** data to determine the size of the direct dynamic degrees of freedom including extra points. If extra points exist, the normal modes and the unsteady spline matrix (input in the analysis set) are expanded to include the extra point degrees

of freedom. The module then computes the **PHIKH** matrix of structural mode shapes splined to the aerodynamic degrees of freedom. **QHLLGEN** then calls the **PRUNMK** utility to prepare the **UNMK** data for the discipline dependent unsteady aerodynamic matrices. The total number of m-k/symmetry sets associated with the **QKK** matrix are computed and the requisite memory for the subsequent computations is obtained. The module then proceeds with the premultiplication of the **QKK** matrix list by the **PHIKH** matrix:

$$[QHKL] = [PHIKH][QKKL]$$

The **QHLL** output matrix is then flushed and computed using one of two paths. If there is only one m-k/symmetry set (which is very rare), the **QHLL** matrix may be formed by a post-multiplication of **QHKL** in one step. If more than one matrix is in the **QHKL** matrix list, however, the module extracts each matrix individually using the **EXQKK** utility and performs the multiplication:

$$[QZHH] = [QHK][PHIKH]$$

and appends the resultant matrix onto **QHLL**.

The matrix **QHJL** is also output. Since this matrix only requires a premultiplication of the input **QKJL** matrix list by **PHIKH**, it is performed in one step and the module terminates.

Design Requirements:

1. The **UNSTEADY** module must have been executed to generate the aerodynamic matrices and generate the **UNMK** entity.

Error Conditions:

None

Engineering Application Module: **RBCHECK**

Entry Point: **RDGCHK**

Purpose:

To compute the rigid body strain energies associated with displacements of each support degree of freedom.

MAPOL Calling Sequence:

```
CALL RBCHECK ( BCID, USET(BC), BGPDT(BC), [D(BC)], [KLL], [KRR], [KLR] );
```

BCID	User defined boundary condition identification number (Integer, Input)
USET(BC)	The unstructured entity defining structural sets (Input), where **BC** represents the MAPOL boundary condition loop index number
BGPDT(BC)	Relation containing basic grid point coordinate data (Input), where **BC** represents the MAPOL boundary condition loop index number
[D(BC)]	Rigid body transformation matrix (Input), where **BC** represents the MAPOL boundary condition loop index number
[KLL]	The stiffness matrix in the l-set degrees of freedom (Input)
[KRR]	The stiffness matrix in the r-set degrees of freedom (Input)
[KLR]	The off-diagonal l-r partition of the a-set stiffness matrix (Input)

Application Calling Sequence:

None

Method:

The **RBCHECK** module begins by checking if the **USET** entity contains any support (r-set) degrees of freedom. If not, the module returns. The module continues by reading the **BGPDT** into memory and then computing the strain energy associated with the rigid body displacements:

$$\mathbf{[X]} = \mathbf{[KLR]}^{T}\ \mathbf{[D]} + \mathbf{[KRR]}$$

The **X** and **KRR** matrices are then read into memory and two normalization measures are computed. The first is the overall norm of each matrix:

$$X_{norm} = \sum_{i=1}^{nr} \sum_{j=1}^{nr} | X_{ij} |$$

$$\mathbf{KRR}_{norm} = \sum_{i=1}^{nr} \sum_{j=1}^{nr} | \mathbf{KRR}_{ij} |$$

$$\varepsilon_{matrix} = \frac{X_{norm}}{\mathbf{KRR}_{norm}}$$

The second is the norm of each of the *nr* columns:

$$X_{j_{norm}} = \sum_{i=1}^{nr} | X_{ij} |$$

$$\mathbf{KRR}_{j_{norm}} = \sum_{i=1}^{nr} | \mathbf{KRR}_{ij} |$$

$$\varepsilon_{col} = \frac{X_{j_{norm}}}{\mathbf{KRR}_{j_{norm}}}$$

These error ratios and norms are then printed out along with the associated diagonal of **x** (the strain energy) for each support degree of freedom.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **RECOVA**

Entry Point: **RECOVA**

Purpose:

To recover the symmetric or asymmetric f-set displacements or accelerations if there are omitted degrees of freedom.

MAPOL Calling Sequence:

```
CALL RECOVA ( [UA], [PO], [GSUBO(BC)], NRSET, [AA], [IFM(BC)], SYM,
              [KOOINV(BC)], [KOOU(BC)], [PFOA(BC)], [UF] );
```

[UA]	Matrix of displacements or accelerations in the analysis set (Input)
[PO]	Optional matrix of static loads applied to omitted degrees of freedom (Input)
[GSUBO(BC)]	Static condensation transformation matrix (Input), where **BC** represents the MAPOL boundary condition loop index number
NRSET	Flag indicating that inertia relief effects are to be included (Integer, Input)
[AA]	Optional matrix of analysis set accelerations for inertia relief (Input)
[IFM(BC)]	Optional matrix containing terms needed for inertia relief (Input), where **BC** represents the MAPOL boundary condition loop index number
SYM	Optional symmetry flag; **=1** if any **KFF** is not symmetric (Integer, Input)
[KOOINV(BC)]	Matrix containing the inverse of **KOO** for symmetric stiffness matrices or the lower triangular factor of **KOO** for asymmetric matrices (Input), where **BC** represents the MAPOL boundary condition loop index number
[KOOU(BC)]	Optional matrix containing the upper triangular factor of **KOO** for asymmetric stiffness matrices (Input), where **BC** represents the MAPOL boundary condition loop index number
[PFOA(BC)]	The partitioning vector splitting the free degrees of freedom into the analysis set and the omitted degrees of freedom (Input), where **BC** represents the MAPOL boundary condition loop index number
[UF]	Matrix containing the displacements or accelerations for the free degrees of freedom (Output)

Application Calling Sequence:

None

Method:

The **RECOVA** module begins by checking if the **PO** argument is nonblank. If so, the displacements at the omitted degrees of freedom due to the loads at the omitted degrees of freedom, **UOO**, are computed. These computations depend on whether inertia relief and/or asymmetric stiffnesses exist. If inertia relief is required (**NRSET > 0**) the loads on the omitted DOF's are modified using the **IFM** matrix and the analysis set accelerations, **AA**; both of which must be input:

```
[PO] = [PO] - [IFM] [AA]
```

The **UOO** terms are then computed from the inverted **KOO** terms based on the **SYM** flag; with the symmetry flag indicating whether the general or symmetric forward backward substitution is used:

$$[UOO] = [KOO]^{-1}[PO] \text{ using Forward Backward Substitution}$$

Finally, the omitted displacements, **UO**, are computed from:

$$[UO] = [GSUBO][UA] + [UOO]$$

Note that the module assumes that the correct set of **KOOINV, KOOU**, **IFM**, **AA**, and **PO** matrices are supplied to match the **SYM** and **NRSET** flags. If the **PO** argument is omitted from the calling sequence, the **UO** terms are computed directly from:

$$[UO] = [GSUBO][UA]$$

with the **GSUBO** argument required to perform the computation. Note that these computations are the same irrespective of the **NRSET** flag. When **UO** is complete, the module merges the computed **UO** terms with the supplied **UA** terms to form the **UF** output.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **SAERO**

Entry Point: **SAERO**

Purpose:

To solve the trim equation for steady aeroelastic trim analyses and to compute the rigid and flexible stability coefficients for steady aeroelastic analyses and the aerodynamic effectiveness constraints for constrained optimization steady aerodynamic analyses.

MAPOL Calling Sequence:

```
CALL SAERO ( NITER, BCID, MINDEX, SUB, SYM, QDP, STABCF, BGPDT(BC),
             [LHSA(BC,SUB)], [RHSA(BC,SUB)], [AAR], [DELTA(SUB)], [PRIGID],
             [R33], CONST, AEFLG(SUB), [AARC], [DELC] );
```

NITER	Design iteration number (Integer, Input)
BCID	User defined boundary condition identification number (Integer, Input)
MINDEX	Mach number index for the current subscript value. (Integer, Input)
SUB	Subscript number of **SAERO** subcases considered in this call (Integer, Input)
SYM	The symmetry flag for the current **SAERO** subcases (Integer, Input)
QDP	Dynamic pressure associated with the current subscript (Real, Input)
STABCF	Relation of rigid stability coefficient data (Input)
BGPDT(BC)	Relation of basic grid point coordinate data (Input), where **BC** represents the MAPOL boundary condition loop index number
[LHSA(BC,SUB)]	Matrix of modified inertia coefficients (Input), where **BC** represents the MAPOL boundary condition loop index number
[RHSA(BC,SUB)]	Matrix of applied load vectors reduced to the r-set (Input), where **BC** represents the MAPOL boundary condition loop index number
[AAR]	Matrix of acceleration vectors (Output)
[DELTA(SUB)]	Matrix of configuration parameters (Output)
[PRIGID]	Rigid load matrix (Input)
[R33]	Reduced rigid body mass matrix (Input)
CONST	Relation of constraint values (Input)
AEFLG(BC)	The logical flag denoting presence of aeroelastic constraints (Logical, Output), where **BC** represents the MAPOL boundary condition loop index number
[AARC]	Matrix of structural accelerations due to unit configuration parameters for use in sensitivity evaluation (output)
[DELC]	Matrix of "unit" flight configuration parameters used to generate the **AARC** accelerations (output)

Application Calling Sequence:

None

Method:

The module begins by bringing into memory the **CASE** entries associated with **SAERO** subcases in the current boundary condition. Then, the **STABCF** relation is read into memory for the current **MINDEX** value. The **TRIM** relation is read for all entries that have the current subcript value and other trim data from **AEROS**, **CONEFFS**, and **CONLINK** are also read into memory.

Then an evaluation of the trim data is done to determine the number of trim subcases that will be solved during this pass (for the current subscript). The **AROCHK** utility is used to evaluate the **SUPORT** condition to ensure (again) that it satisfies the requirements of the **TRIM** solver and to get the names and DOFs of the supported degrees of freedom. Then, after creating needed scratch entities, the grand loop on the trim subcases begins.

Each trim subcase must be solved separately because of the options for control effectiveness and control linking. The first step is to determine which **TRIM** entries are associated with the current subcase (note all are associated with the current subscript). Once the **TRIM** id of the current case is known, the **CASE** relation data are searched to determine the subcase number (1 to n over all **SAERO** entries in **CASE** for each **BC**). Then the **AROLNK** routine is called to assemble a linking matrix of control effectiveness factors and linking relationships for the current subscript such that:

$$\{\delta\} = [\mathbf{TLINK}] * \mathbf{DELRED}$$

where the **DELRED** matrix is reduced to only the active trim parameters and the effectiveness factors have been included. Then the rigid and flexible loads are hit with the linking matrix to reduce the problem to the relevant configuration parameters:

$$\mathbf{P2RED} = \mathbf{P2} * \mathbf{TLINK}$$

$$\mathbf{RHSRED} = \mathbf{RHS} * \mathbf{TLINK}$$

P2RED and **RHSRED** contain one row for each structural acceleration and one column for each label on the trim entry This means that the total number of stability parameters (either fixed or free) is the number of columns in **P2** and **RHS**. Further, the order of the parameters is the order given on the **TRIM** tuples.

Now the trim equations can be assembled. From the input, we have the relationship

$$\begin{bmatrix} \mathbf{LHS}_{ff} & \mathbf{LHS}_{fk} \\ \mathbf{LHS}_{kf} & \mathbf{LHS}_{kk} \end{bmatrix} \begin{bmatrix} \mathbf{AR}_{free} \\ \mathbf{AR}_{known} \end{bmatrix} = \begin{bmatrix} \mathbf{RHS}_{fu} & \mathbf{RHS}_{fs} \\ \mathbf{RHS}_{ku} & \mathbf{RHS}_{ks} \end{bmatrix} \begin{bmatrix} \mathbf{DEL}_{u} \\ \mathbf{DEL}_{s} \end{bmatrix}$$

Where:	Represents:
F+K	Number of **SUPORT** point DOFs
F	Set of free accelerations, **AR**
K	Set of known(**FIXED**) accelerations, **AR**
U+S	Number of **AERO** parameters
U	Set of unknown parameters
S	Set of set(**FIXED**) parameters

These equations must be rearranged to get free accelerations and unknown delta's on the same side of the equation:

$$\begin{bmatrix} \mathbf{LHS}_{ff} & -\mathbf{RHS}_{fu} \\ \mathbf{LHS}_{uf} & -\mathbf{RHS}_{uu} \end{bmatrix} \begin{bmatrix} \mathbf{AR}_{free} \\ \mathbf{DEL}_{u} \end{bmatrix} = \begin{bmatrix} -\mathbf{LHS}_{kk} & \mathbf{RHS}_{ks} \\ -\mathbf{LHS}_{sk} & \mathbf{RHS}_{ss} \end{bmatrix} \begin{bmatrix} \mathbf{AR}_{k} \\ \mathbf{DEL}_{s} \end{bmatrix}$$

and we must handle the degenerate case where all accelerations or all delta's are known.

Following rearrangement of the equations, the unknowns are solved for in the ARTRMS/D routine. First the rigid masses and loads, **P2RED** and **MRR** are used to obtain the rigid trim and then the flexible inputs **RHSRED** and **LHS** are used for the "real" solution.

Then, the flexible results are unscrambled and the rigid body accelerations (either input on the **TRIM** or output from the solution of the above) are stored on the **AAR** matrix and the same is done with the trim parameters after the **TLINK** matrix is used to recover the full vector from the reduced set. Then the results for the rigid and flexible trim are printed.

Only if the print is requested or if constraints are applied are the stability coefficients computed. These data are recomputed in each subcase because the effectiveness terms affect the stability derivative outputs. The **ARSCFS/D** module is called to compute the flexible data from the forces on the support degrees of freedom due to the unit configuration parameters:

$$[F] = [MRR][LHS]^{-1}[RHS]$$

The **P2** matrix contains the same information for the rigid aerodynamic loads (computed in the MAPOL sequence). These data are then normalized and the stability coefficient table stored into memory. Once complete, the stability coefficient table is printed using the effectiveness factors and linking terms to assemble the "dependent" coefficients and factor all coefficients according to the user input.

Finally, using the in-core table of derivatives, the **ARCONS/D** submodule is called to evaluate the constraints for the current subcase. These constraints are evaluated from the stability coefficient table but, to prepare for eventual sensitivity computations, the additional outputs **AEFLG**, **AARC** and **DELC** are needed. The first is a logical flag to indicate to the MAPOL sequence that the **AARC** and **DELC** matrices are full. The **AARC** matrix and **DELC** matrix contain one or more columns for each constraint (appended in the order the constraints are evaluated). The **AARC** contains the accelerations of the support DOFs due to the unit configuration parameter vectors in **DELC**. This pair of matrices will allow the computation of the derivative of the accelerations due to the unit parameters which is an essential ingredient in the sensitivity computation.

For lift effectiveness constraints

AARC - 1 column due to unit **ALPHA**

DELC - 1 column containing a unit **ALPHA** with all others 0.0

For aileron effectiveness constraints

AARC - 2 columns; the first for unit **SURFACE** rotation and the second for unit roll rate (**PRATE**).

DELC - 2 columns containing a unit rotation of the named **SURFACE** and the second a unit **PRATE**

For stability coefficient constraints (**DCONSCF**)

AARC - 1 column due to unit **PARAMETER** where **PARAMETER** is that named on the constraint entry

DELC - 1 column containing a unit **PARAMETER** with all others 0.0

DCONTRM are evaluated at this time, but do not require any pseudodisplacements for sensitivity evaluation. The pseudodisplacements are those which arise due to the unit accelerations that arise due to unit configuration parameters.

After the stability coefficients (and constraints) are computed and printed, the rigid and flexible trim results are printed and the module repeats the entire process for all the subcases that are associated with the current **SUB**script. Then the module terminates.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **SAERODRV**

Entry Point: **SARODR**

Purpose:

MAPOL director for steady aeroelastic analyses.

MAPOL Calling Sequence:

```
CALL SAERODRV (BCID, SUB, LOOP, MINDEX, SYM, MACH, QDP, PRINT );
```

BCID	User defined boundary condition identification number (Integer, Input)
SUB	Current Mach number subscript number (Integer, Input)
LOOP	Logical flag indicating whether another subscript is required to complete the set of all subcases (Logical, Output)
MINDEX	Mach number index associated with the current subscript. (Integer, Output)
SYM	**SYM**metry flag for the current subscript. (Integer, Output) = **1** symmetric = **-1** Antisymmetric
MACH	Mach number associated with the current subscript (Real, Output)
QDP	Dynamic pressure associated with the current subscript (Real, Output)
PRINT	Optional print flag indicating that the summary of trim cases associated with the current pass (subscript) is to be printed to the standard output. (In the standard sequence, **PRINT** is used only during analysis not during sensitivity analysis) (Optional, Integer, Input)

Application Calling Sequence:

None

Method:

First the **CASE** relation is read to determine the **TRIM** ids and **SYM**metries of all **SAERO** cases in the current boundary condition. If any exist, the **TRIM** relation is opened and read into memory. Each trim entry referenced in **CASE** is then compressed into a format containing the **TRIM** id, Mach number, dynamic pressure, trim type, Mach number index, subscript and subcase id.

Once these data are collected, the **CASE** tuples read into memory are looped over to choose which **TRIM** cases are to be analyzed for this subscript value. There are four steps in choosing the proper trim cases:

(1) Take the first **SAERO** subcase in **CASE** that has not been done on an earlier pass — cases already analyzed will reference trims with a "subscript" value that is not "null" (uninitialized) and that is less than the current value of **SUB** — on the first design iteration all subscript values will be "null"

(2) Once the parent case is known, choose that case and all others with the same Mach, **QDP** and **TRMTYP**

(3) Update the "subcript" attribute in **TRIM** to mark all the cases that are being processed. Also load the **SUBID** to assist in re-merging the answers into **CASE** subcase order

(4) Check if any more saero subcases need to be processed and set the "loop" flag

After these steps have been completed, if the **PRINT** flag is nonzero, a summary of the selected **TRIM**s is printed to the output file.

Design Requirements:

1. The **TRIM** relation is assumed to contain **NULL** values for **SUBSCRPT** on the first subscript of the first design iteration (for **OPTIMIZE** boundary conditions) and for the first subscript of all **ANALYZE** boundary conditions.

Error Conditions:

None

Engineering Application Module: **SAEROMRG**

Entry Point: **SAROMR**

Purpose:

Merges the static aero results for each subscript (stored in the matrix **[MATSUB]**) into the **[MATOUT]** matrix in case order rather than subscript order for the **BC**'th boundary condition.

MAPOL Calling Sequence:

```
CALL SAEROMRG ( BCID, SUB, [MATOUT], [MATSUB] );
```

BCID	User defined boundary condition identification number (Integer, Input)
SUB	Current Mach number subscript (Input, Integer)
[MATOUT]	Merged output matrix reordered to be in **CASE** order for the current boundary condition (Input and Output)
[MATSUB]	Generic input matrix containing data for the current subscript value in **TRIM** id order of **TRIM** cases associated with the current subscript (Input)

Application Calling Sequence:

None

Method:

First the **CASE** relation is read to retrieve the trim id's for the **SAERO** subcases in the current boundary condition. The the **TRIM** relation is read to obtain the subcase numbers associated with each trim id having the current **SUB**script value.

Then the **MATSUB** and **MATOUT** matrices are opened. If **MATOUT** is uninitialized ***or*** if **SUB = 1**, it is initialized (flushed and the number of rows, precision and form set to those of **MATSUB**. If **MATOUT** already exists and has data in it, a scratch matrix is created to hold the final merged data.

For each **SAERO CASE** entry for the current boundary, the **TRIM** data are searched to determine the subscript number associated with the subcase. If the subscript is less than **SUB**, a column from **MATOUT** will be taken (it was stored there on an earlier pass). If the subscript is equal to **SUB**, it will be stored on the output matrix from **MATSUB**. If greater than **SUB**, it is ignored till later passes.

Once a column is identified as active in **MATSUB** (**PGAA** indicates active and subscript = **SUB**), an additional check is made to see if the column is active in **PGUA**. Only those columns that are active in **PGUA** are copied to **MATOUT**. This filtering is done to limit the amount of computational effort in the stress, strain and displacement constraint sensitivity computations that proceed using the **MATOUT** matrix. The **MATSUB** columns that are active due to **DCONTRM** constraints are no longer needed as these sensitivities are assumed to have been computed already in the **AEROSENS** module.

Once the final matrix is formed, if **MATOUT** had had data in it, the name of the scratch matrix that was loaded is switched with that of **MATOUT**. The scratch entity is then destroyed.

Design Requirements:

1. The assumption is that each **MATSUB** matrix contains the results from the "**SUB**"th subscript value in the order the trim id's for that **SUB** appear in the **TRIM** relation.

2. The same **MATOUT** matrix must be passed into the **AROSNSMR** module on each call since the columns associated with earlier subscript values are read from **MATOUT** into a scratch entity. The merged matrix that results then replaces the input **MATOUT**.

3. The **AEROSENS** module is called upstream of the **AROSNSMR** module to process active **DCONTRM** constraints for the current subscript. Thus, those columns that are active only for **DCONTRM** constraints may be filtered out for the downstream processing of stress, strain and displacement constraints.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **SCEVAL**

Entry Point: **SCEVAL**

Purpose:

To compute the stress and/or strain constraint values for the statics or steady aeroelastic trim analyses in the current boundary condition.

MAPOL Calling Sequence:

```
CALL SCEVAL ( NITER, BCID, [UG(BC)], [SMAT], [NLSMAT], SMATCOL, NLSMTCOL,
              TREF, TREFD, [GLBSIG],[NLGLBSIG], CONST, DSCFLG );
```

NITER	Design iteration number (Integer, Input)
BCID	User defined boundary condition identification number (Integer, Input)
[UG(BC)]	The matrix of global displacements for all static applied loads in the current boundary condition (Input), where **BC** represents the MAPOL boundary condition loop index number.
[SMAT]	Matrix entity containing the linear portion of the sensitivity of the stress and strain components to the global displacements (Input)
[NLSMAT]	Matrix entity containing the nonlinear portion of the sensitivity of the stress and strain components to the global displacements (Input)
SMATCOL	Relation containing matrix **SMAT** column information (Character,Input)
NLSMTCOL	Relation containing matrix **NLSMAT** column information (Character,Input)
TREF	Unstructured entity containing the linearly designed element reference temperatures (Input)
TREFD	Unstructured entity containing the nonlinearly designed variable element reference temperatures (Input)
[GLBSIG]	Matrix of stress/strain components for all the applied linearly designed stress constraints for the current boundary condition (Output)
[NLGLBSIG]	Matrix of stress/strain components for all the applied nonlinearly designed stress constraints for the current boundary condition (Output)
CONST	Relation of constraint values (Const)
DSCFLG	The discipline flag (Integer, Input) = **0** statics > **0** static aeroelasticity

Application Calling Sequence:

None

Method:

The **SCEVAL** module begins by determining if there are any stress constraints applied and any user functions which require element response functions. If any are found, execution continues.

First the **CASE** relation is read. Then, if the call is associated with **SAERO** disciplines, the **TRIM** relation is read to associate, for each subcase, the subcase id and the subscript id. Then an in-core table is formed that contains, for the subcases in this boundary condition the **DISFLAG**, **SUBSCRIPT**, and **THERMID**. The

latter is for thermal load corrections to the stresses and strains. If any thermal load cases were found, the **GRIDTEMP** and **TREF** entities are opened.

If the current boundary condition is the first with stress or strain constraints, the running constraint type count variables are reinitialized for the current design iteration. This type count provides a link between the **ACTCON** print of design constraints and the debug print option supported by the **SCEVAL** module. If any thermal loads exist for the current boundary condition, the **GRIDTEMP**, **TREF** and **TREFD** entities are brought into memory to be available for the computation of the stress-free thermal strain correction to the element stresses. Once these preparations have been made, the **SMAT** and **NLSMAT** matrices of stress/strain sensitivities and the **GLBSIG** and **NLGLBSIG** matrices are opened and the **GLBSIG** and **NLGLBSIG** matrices are positioned to the proper columns to pack additional stress/strain components. Note that the **GLBSIG** and **NLGLBSIG** matrices store all the columns associated with the current boundary condition since they are required for the constraint sensitivity computations.

Finally, the **UG** matrix of global displacements is opened. For each column in the **UG** matrix, the matrix products

[GMA] = [SMAT]{UG} and **[NLGMA] = [NLSMAT]{UG}**

are calculated to obtain the component stress or strain values for linearly designed elements and nonlinearly designed elements, respectively. Having calculated and stored in core these values, the element dependent constraint evaluation routines are called to process each constraint. Note that the order in which the element routines are called must be the same as the order the **SMAT** and **NLSMAT** columns were formed. That order is:

1. Bar elements, **BARSC** (Using both **[GMA]** and **[NLGMA]**)
2. Isoparametric quadrilateral membrane elements, **QD1SC** (Using **[GMA]** only)
3. Quadrilateral bending plate elements, **QD4SC** (Using both **[GMA]** and **[NLGMA]**)
4. Rod elements, **RODSC** (Using **[GMA]** only)
5. Shear panels, **SHRSC** (Using **[GMA]** only)
6. Triangular bending plate elements, **TR3SC** (Using both **[GMA]** and **[NLGMA]**)
7. Triangular membrane elements, **TRMSC** (Using **[GMA]** only)

On the first pass through the element dependent routines, all the **xxxxEST** tuples (i.e., **RODEST** and **TRMEMEST**) with nonzero stress/strain constraint flags are retrieved from the data base. For subsequent passes, this information is used directly from core. Each constraint is evaluated in turn with the stress components modified by the thermal stress correction if the displacement field includes thermal strain effects. The **CONST** relation is loaded with one tuple for each constraint as they are processed. When all the constraints have been evaluated for the current loading condition, the adjusted linear design variable and nonlinear design variable stress/strain constraint terms are packed to the **GLBSIG** and **NLGLBSIG** matrices.

The element stress and strain responses which are required by any user function constraints are also computed in this module. Those response values are stored into a relation entity to be used by user function evaluation utilities.

Design Requirements:

1. The **SMAT** (or **NLSMAT**), **GRIDTEMP** and **TREF** (or **TREFD**) entities must exist.
2. The **CASE** relation must be complete from **SOLUTION**.

Error Conditions:

1. A zero material allowable may cause division by zero in the computation of some of the constraints.

Engineering Application Module: **SOLUTION**

Entry Point: **SOLUTION**

Purpose:

To interpret the solution control packet.

MAPOL Calling Sequence:

```
CALL SOLUTION ( NUMOPTBC, NBNDCOND, K6ROT, MPS, MPE, OCS, OCE, FSDS, FSDE,
                MAXITER, MOVLIM, WINDOW, OCMOVLIM, ALPHA,CNVRGLIM,
                NRFAC, EPS, FDSTEP );
```

NUMOPTBC	Number of optimization boundary conditions (Integer, Output)
NBNDCOND	Total number of optimization and analysis boundary conditions (Integer, Output)
K6ROT	Stiffness value for plate element "drilling" degrees of freedom (Real,Output)
MPS	The first iteration to use math programming (Integer, Output)
MPE	The last iteration to use math programming (Integer, Output)
OCS	The first iteration to use optimality criteria (Integer, Output)
OCE	The last iteration to use optimality criteria (Integer, Output)
FSDS	The first iteration to use **FSD** (Integer, Output)
FSDE	The last iteration to use **FSD** (Integer, Output)
MAXITER	The maximum number of allowable iterations (Integer, Output)
MOVLIM	Limit on how much a design variable can move for this iteration in using math programming (Real, Output)
WINDOW	The window around the zero in which the **MOVLIM** bound is overridden to allow the local variable to change sign. If **WINDOW = 0.0**, the local variable may not change sign. If **WINDOW** is nonzero, the half width of a band around zero, **EPS** is computed `EPS = WINDOW/100 * MAX ( ABS(TMIN), ABS(TMIN) )` If the local variable falls within the band, the new minimum or maximum for the current iteration is changed to lie on the other side of zero from the local variable. The bandwidth **EPS** is a percentage of the larger of **TMAX** or **TMIN** where **WINDOW** specifies the percentage. (Real, Output)
OCMOVLIM	Limit on how much a design variable can move for this iteration in using optimality critera (Real, Output)
ALPHA	Exponential move limit for the **FSD** algorithm (Real, Output)
CNVRGLIM	Relative percent change in the objective function that indicates approximate problem convergence (Real, Output)
NRFAC	Determines the minimum number of retained constraints equal to **NRFAC*NDV** (Real, Output)

EPS	A second criteria for constraint retention. All constraints greater than or equal to **EPS** will be retained (Real, Output)
FDSTEP	Relative design variable increment for finite difference computations (Real,Output)

Application Calling Sequence:

None

Method:

The **SOLUTION** module interprets the solution control statements and loads the resultant information to the **CASE** relation. On completion of the routine, the total number of all boundary conditions, the number of analysis boundary conditions and the user's optimization strategy are output to the executive sequence to direct the MAPOL execution path.

Design Requirements:

1. A Solution Control packet must be included in the input data stream.

Error Conditions:

1. Syntax errors and inconsistent or illegal solution control requests are flagged and the execution is terminated.

Engineering Application Module: **SPLINES**

Entry Point: **SPLINE**

Purpose:

Generates the interpolation matrix that relate displacements and forces between the structural and steady aerodynamic **MODEL**s.

MAPOL Calling Sequence:

```
CALL SPLINES ( GSIZEB, GEOMSA, AECOMPS, AEROS, [GTKG], [GSTKG] );
```

GSIZEB	The number of degrees of freedom in the set of all structural GRID and SCALAR points (Integer, Input)
GEOMSA	A relation describing the aerodynamic boxes for the steady aerodynamics model. The location of the box centroid, normal and pitch moment axis are given. It is used in splining the aerodynamics to the structure and to map responses back to the aerodynamic boxes (Input)
AECOMPS	A relation describing aerodynamic components for the steady aerodynamics model. It is used in splining the aerodynamics to the structural model (Input)
AEROS	A relation containing the definition of the aerodynamic coordinate system (Input)
[GTKG]	The matrix of splining coefficients relating the aerodynamic pressures to forces at the structural grids (Output)
[GSTKG]	The matrix of splining coefficients relating the structural displacements to the streamwise slopes of the aerodynamic boxes (Output)

Application Calling Sequence:

None

Method:

All the **SPLINE1**, **SPLINE2** and **ATTACH** data are read and those associated with the steady aerodynamic model as described by the **AECOMPS** entity are used to assemble a list of aerodynamic boxes and structural grids for each spline. The **GEOMSA** relation is used to obtain the basic coordinates of the aerodynamic boxes and the **BGPDT** relation is used to obtain the locations of the structural grids. The spline matrix consisting of two columns (displacement and slope) for each aerodynamic box and 6 rows for each structural grid is then assembled for the aerodynamic boxes and structural grids attached to the spline.

The spline matrix is then expanded to include two columns for each aerodynamic box in the steady aerodynamic model and **GSIZEB** rows. It is then split into two pieces with each odd-numbered column (displacement) merged with previously processed splines to form the **GTKG** matrix and each even numbered (slope) column merged to form **GSTKG**. The process is repeated until all splines have been completed. The final matrices are returned to the MAPOL sequence.

Design Requirements:

None

Error Conditions:

1. Each aerodynamic box may appear on only one **SPLINE1**, **SPLINE2** or **ATTACH** entry although not all boxes need appear. Missing boxes will not influence the aeroelastic response.

2. Missing structural grids or aerodynamic elements appearing on the spline definitions will be flagged.

Engineering Application Module: **SPLINEU**

Entry Point: **SPLINE**

Purpose:

Generates the interpolation matrix that relate displacements and forces between the structural and unsteady aerodynamic **MODEL**s.

MAPOL Calling Sequence:

```
CALL SPLINEU ( GSIZEB, GEOMUA, AECOMPU, AERO, [UGTKG] );
```

GSIZEB	The number of degrees of freedom in the set of all structural GRID and SCALAR points (Integer, Input)
GEOMUA	A relation describing the aerodynamic boxes for the unsteady aerodynamics model. The location of the box centroid, normal and pitch moment axis are given. It is used in splining the aerodynamics to the structure and to map responses back to the aerodynamic boxes (Input)
AECOMPU	A relation describing aerodynamic components for the unsteady aerodynamics model. It is used in splining the aerodynamics to the structural model (Input)
AERO	A relation containing the definition of the aerodynamic coordinate system (Input)
[UGTKG]	The matrix of splining coefficients relating the aerodynamic pressures to forces at the structural grids and relating the structural displacements to the streamwise slopes of the aerodynamic boxes (Output)

Application Calling Sequence:

None

Method:

All the **SPLINE1**, **SPLINE2** and **ATTACH** data are read and those associated with the unsteady aerodynamic model as described by the **AECOMPU** entity are used to assemble a list of aerodynamic boxes and structural grids for each spline. The **GEOMUA** relation is used to obtain the basic coordinates of the aerodynamic boxes and the **BGPDT** relation is used to obtain the locations of the structural grids. The spline matrix consisting of two columns (displacement and slope) for each aerodynamic box and 6 rows for each structural grid is then assembled for the aerodynamic boxes and structural grids attached to the spline.

The spline matrix is then expanded to include two columns for each aerodynamic box in the unsteady aerodynamic model and **GSIZEB** rows. It is then merged with previously processed splines. The process is repeated until all splines have been completed. The final **[UGTKG]** matrix is returned to the MAPOL sequence.

Design Requirements:

None

Error Conditions:

1. Each aerodynamic box may appear on only one **SPLINE1**, **SPLINE2** or **ATTACH** entry although not all boxes need appear. Missing boxes will not influence the aeroelastic response.

2. Missing structural grids or aerodynamic elements appearing on the spline definitions will be flagged.

Engineering Application Module: **STEADY**

Entry Point: **STEADY**

Purpose:

To perform preface aerodynamic processing for planar steady aerodynamics.

MAPOL Calling Sequence:

```
CALL STEADY ( MINDEX, TRIMDATA, AECOMPS, GEOMSA, STABCF, [AICMAT(MINDEX)],
              [AAICMAT(MINDEX)], [AIRFRC(MINDEX)], AEROGEOM, CAROGEOM );
```

MINDEX	Mach number index for the current pass. Controls which Mach Number/symmetry conditions will be processed in this pass of **STEADY**. One pass for each unique Mach number will be performed with **MINDEX** incrementing by one until **TRIMCHEK** returns **LOOP=FALSE** (Input)
TRIMDATA	A relation created by **TRIMCHEK** that contains the description of the **TRIM** entries for each boundary condition and each subcase. Additional subscripts have been added to the **TRIM** data to associate Mach number values with **MINDEX** subscripts and the input accelerations have been normalized to be in consistent units (Input)
AECOMPS	A relation describing aerodynamic components for the planar **STEADY** aerodynamics **MODEL**. It is used in splining the aerodynamics to the structural model (Output)
GEOMSA	A relation describing the aerodynamic boxes for the planar **STEADY** aerodynamics **MODEL**. The location of the box centroid, normal and pitch moment axis are given. It is used in splining the aerodynamics to the structure and to map responses back to the aerodynamic boxes (Output)
STABCF	A relation of rigid stability coefficients for unit configuration parameters. The rigid coefficients are stored in **STABCF** and the corresponding distributed forces are stored in **AIRFRC**. The **STABCF** relation is used to pick the appropriate rigid loads from **AIRFRC** when performing the aeroelastic trim as well as for retrieving the **RIGID/DIRECT** stability coefficients for each configuration parameter (Output)
[AICMAT(MINDEX)]	Matrix containing the **STEADY** aerodynamic influence coefficients for **SYM**metric Mach numbers (Output)
[AAICMAT(MINDEX)]	Matrix containing the **STEADY** aerodynamic influence coefficients for anti**SYM**metric Mach numbers (Output)
[AIRFRC(MINDEX)]	Matrix containing the aerodynamic forces for unit configuration parameters for the current Mach number index. If both **SYM**metric and anti**SYM**metric conditions exist for the Mach number, both sets of configuration parameters will coexist in **AIRFRC** (Output)
AEROGEOM	An aerodynamic geometry relation output only for geometry checking. The "grids" defined in **AEROGEOM** are "connected" to 2-node (RODs) and 4-node (**QUAD**s) elements in the **CAROGEOM** in such a way as to emulate the structural **MODEL**. ICE may then be used to punch an equivalent structural **MODEL** to allow graphical presentation of the **STEADY** aero model

CAROGEOM	A aerodynamic geometry relation output only for geometry checking. The "grids" defined in **AEROGEOM** are "connected" to 2-node (**RODs**) and 4-node (**QUADs**) elements in the **CAROGEOM** in such a way as to emulate the structural **MODEL**. ICE may then be used to punch an equivalent structural model to allow graphical presentation of the **STEADY** aero model

Application Calling Sequence:

None

Method:

The **STEADY** preface module performs initial aerodynamic processing for planar **STEADY** aerodynamics. It is driven by the the **TRIMDATA** relation and the **MINDEX** value.

On each call, the **TRIMDATA** relation is queried to determine the **MINDEX**'th Mach number and whether symmetric, antisymmetric or both boundary conditions are to be applied.

On the first call (determined by **MINDEX=1**) the **STEADY** module computes the planar **STEADY** aerodynamic geometry in calls to **GEOM**. It then processes the current Mach number and stores the resultant **AIC** terms in the **AICMAT** and/or **AAICMAT** entity (depending on the symmetry options) and in the resultant rigid forces in the **AIRFRC** matrix. The **STABCF** relation is loaded for the current **MINDEX** value with the symmetric and antisymmetric stability derivatives in the same order that the **AIRFRC** matrix columns are loaded. Hence, the **STABCF** relation points to the corresponding **AIRFRC** column.

Design Requirements:

1. The **STEADY** module interacts with the executive in that the **MINDEX** parameter should be unique for each call (although it need not be monotonically increasing). The **MINDEX** value must be 1 on the first call to ensure that the geometry processing is done.

Error Conditions:

1. Errors in the **STEADY** aerodynamic **MODEL** specifications are flagged.

Engineering Application Module: **STEADYNP**

Entry Point: **STDYNP**

Purpose:

Non-planar **STEADY** aeroelastic analysis preface.

MAPOL Calling Sequence:

```
CALL STEADYNP ( NONPONLY, AECOMPS, GEOMSA, STABCF, [AIRFORCE], AEROGEOM,
                CAROGEOM, OAGRDLOD );
```

NONPONLY	A logical flag returned to the MAPOL sequence that is true if the only disciplines in the **CASE** relation are nonplanar static aerodynamics subcases. If so, the MAPOL sequence should terminate since there is no further steps to be taken (Output)
AECOMPS	A relation describing aerodynamic components for the nonplanar **STEADY** aerodynamics model. (Output)
GEOMSA	A relation describing the aerodynamic boxes for the nonplanar **STEADY** aerodynamics model. The location of the box centroid, normal and pitch moment axis are given (Output)
STABCF	A relation of rigid stability coefficients for unit configuration parameters. The rigid coefficients are stored in **STABCF** and the corresponding distributed forces are stored in **AIRFRC**. The **STABCF** relation is used to pick the appropriate rigid loads from **AIRFRC** when performing the aeroelastic trim as well as for retrieving the **RIGID/DIRECT** stability coefficients for each configuration parameter (Output)
[AIRFORCE]	Matrix containing the aerodynamic forces for unit configuration parameters for the highest Mach number used in the set of **NPSAERO** subcases in the Solution Packet. If both symmetric and antisymmetric conditions exist for the Mach number, both sets of configuration parameters will coexist in **AIRFORCE** (Output)
AEROGEOM	A aerodynamic geometry relation output only for geometry checking. The "grids" defined in **AEROGEOM** are "connected" to 2-node (**RODs**) and 4-node (**QUADs**) elements in the **CAROGEOM** in such a way as to emulate the structural model. ICE may then be used to punch an equivalent structural model to allow graphical presentation of the **STEADY** aero model
CAROGEOM	A aerodynamic geometry relation output only for geometry checking. The "grids" defined in **AEROGEOM** are "connected" to 2-node (**RODs**) and 4-node (**QUADs**) elements in the **CAROGEOM** in such a way as to emulate the structural model. ICE may then be used to punch an equivalent structural model to allow graphical presentation of the **STEADY** aero model
OAGRDLOD	A relation containing the rigid forces and pressures for each **NPSAERO** subcase's set of configuration parameters for the aerodynamic elements whose **PRESSURE** output was requested in Solution Control. These constitute the "trimmed" state of the configuration (Output)

Application Calling Sequence:

None

Method:

The **STEADYNP** preface module perform all the processing for nonplanar **STEADY** aerodynamics. It is driven by the the **TRIM** data present in the bulk data packet and the **NPSAERO** disciplines in the **CASE** relation. The **CASE** relation provides the symmetries while the **TRIM** relation provides the Mach numbers. Only if **NPSAERO** disciplines are in **CASE** is any processing done and both **TRIM** and **AEROS** entries must be found. If only **NPSAERO** cases are found in **CASE**, the **NONPONLY** flag is set to **TRUE**.

Internally, one pass is made over the **STEADYNP** module for each distinct Mach number on the **TRIM**s in **CASE**. On each pass, the **PASDFN** submodule is called to determine the set of all Mach numbers and, for each Mach number, whether symmetric, antisymmetric or both boundary conditions are to be applied. Having determined all unique Mach numbers, the **PASSDF** then determines the Mach number (in numerical order lowest to highest) to be processed. A **MINDEX** value from one to the number of unique Mach numbers is assigned to identify the pass. If the chosen Mach number is the last one, a **LOOP** flag is set to false to tell the **STEADYNP** that no more passes are needed.

On the first pass, the **STEADYNP** module computes the nonplanar **STEADY** aerodynamic geometry in calls to **GEOM**. On the first and subsequenct passes, it then processes the current Mach number and stores the resultant rigid forces due to unit parameters in a scratch entity. A maximum of 50 Mach Index values may be analyzed in a single ASTROS job. The **STABCF** relation is loaded for the current **MINDEX** value with the symmetric and antisymmetric stability derivatives in the same order that the rigid force matrix columns are loaded. Hence, the **STABCF** relation points to the corresponding **AIRFORCE** column.

Once all the **AIRFORCE** matrices for each Mach number index are computed, the **NPATRM** submodule is called to solve the "trim" condition for each subcase. That solution involves assembling the **A** vector from the user's **TRIM** entry and multiplying the rigid unit force matrix by it. If any **PRESSURE** requests are in the Solution Control, the pressures and forces on the appropriate aerodynamic boxes are loaded to the **OAGRDLOD** relation. Finally, the intermediate rigid force matrices are deleted except for the last one. Its name is exchanged for that of **AIRFORCE** so that it is available to the MAPOL sequence for user manipulation.

Design Requirements:

1. A maximum of 50 unique Mach numbers may be analyzed in a single run.

2. Only the last (highest) Mach number's rigid unit loads matrix, **AIRFORCE**, is saved and output to the MAPOL sequence.

Error Conditions:

1. Errors in the **STEADY** aerodynamic **MODEL**s or **TRIM** specifications are flagged.

Engineering Application Module: **TCEVAL**

Entry Point: **TCEVAL**

Purpose:

To compute the current values of thickness constraints for this optimization iteration.

MAPOL Calling Sequence:

```
CALL TCEVAL ( NITER, NDV, MOVLIM, WINDOW, GLBDES, LOCLVAR, [PMINT],
              [PMAXT], [PTRANS], TFIXED, CONST );
```

NITER	Design iteration number (Integer, Input)
NDV	The number of design variables (Integer, Input)
MOVLIM	Move limit to apply to the local design variables (Real, Input): `t/MOVLIM < t < t * MOVLIM; MOVLIM > 1.0`
WINDOW	The window around the zero in which the **MOVLIM** bound is overridden to allow the local variable to change sign. If **WINDOW** = 0.0, the local variable may not change sign. If **WINDOW** is nonzero, the half width of a band around zero, **EPS** is computed `EPS = WINDOW/100 * MAX ( ABS(TMIN), ABS(TMAX) )` If the local variable falls within the band, the new minimum or maximum for the current iteration is changed to lie on the other side of zero from the local variable. The bandwidth **EPS** is a percentage of the larger of **TMAX** or **TMIN** where **WINDOW** specifies the percentage. (Real, Input)
GLBDES	Relation of global design variables (Input)
LOCLVAR	Relation containing the relationship between local variables and global variables in the design problem (Input)
[PMINT]	Matrix entity containing the minimum thickness constraint sensitivities (Input)
[PMAXT]	Matrix entity containing the maximum thickness constraint sensitivities (Input)
[PTRANS]	The design variable linking matrix (Character,Input)
TFIXED	Relation of fixed thickness of layer (Input)
CONST	Relation of constraint values (Output)

Application Calling Sequence:

None

Method:

This module first computes the element thickness functions which are required by any user-defined functional constraints. Those response values are stored into a relational entity to be used by the function evaluation utilities. Then the module determines if any minimum and maximum gauge constraints exist in the problem. These constraints are generated by ASTROS if, and only if, shape function design variable linking is used. If any constraints exist, the vector of design variable values from **GLBDES** and all the **LOCLVAR** data are brought into core. The next step is to determine if any user

specified move limit on the local variables is to be applied to the minimum thickness constraints (note that the maximum thickness constraints are always computed relative to their gauge limits rather than to a move limit).

If move limits are applied (as they almost always are), the **DCONTHK** or **DCONTH2** data are also brought into core to identify which elements minimum gauge constraints are always to be retained by the constraint deletion algorithm in the **ACTCON** module. The minimum gauge constraints are then computed by performing the matrix multiplication:

$$\{g\} = 1.0 - [PMINT]^T\{v\} = 1.0 - \frac{t}{t_{min}}$$

The **LOCLVAR** data is then used to determine to which element each "g" applies. If the constraint value is less critical (more negative) than the move limit value of

$$g_{move} = 1.0 - MOVLIM$$

it is stored on the **CONST** relation as a computed constraint ***only*** if it appears on a **DCONTHK** or **DCONTH2**entry (in which case it will end up as an active constraint from **ACTCON**), otherwise, the constraint is ignored for this design iteration. If the constraint value is more critical than the move limit value, it is only stored on **CONST** if it is on a **DCONTHK** or **DCONTH2** entry ***or*** if the constraint violates a cutoff value set to

$$g_{retain} = 0.10$$

Any minimum thickness constraints that are stored on **CONST** that do not appear on **DCONTHK** or **DCONTH2** entries will be subject to the normal constraint deletion criteria. The maximum gauge constraints are then computed by performing the matrix multiplication:

$$\{g\} = [PMAXT]^T\{v\} + \{v\}_{fixed} - 1.0$$

The **LOCLVAR** data is then used to determine to which element each "g" applies. No move limits are applied to these constraints and they are stored directly to the **CONST** relation to undergo the normal constraint deletion in **ACTCON**.

Design Requirements:

1. This module should be the first module called in the optimization phase of the MAPOL sequence.

2. The move limit that is passed into this routine ***must*** match the value used to evaluate the constraints in the **MAKDFV** module. If not, the constraint sensitivities will be in error with no warning given.

Error Conditions:

1. A local variable has become negative due to insufficient **DCONTHK** or **DCONTH2** entries or illegal gauge constraints.

Engineering Application Module: **TRIMCHEK**

Entry Point: **TRMCHK**

Purpose:

To perform preface aerodynamic processing on the requested **SAERO** discipline requests and their referenced **TRIM** Bulk Data entries.

MAPOL Calling Sequence:

```
CALL TRIMCHEK ( MINDEX, LOOP, GOAERO, CASE, TRIMDATA );
```

MINDEX	Mach number index for the current pass. Controls which Mach Number/symmetry conditions will be processed in this pass of **STEADY**. One pass for each unique Mach number will be performed with **MINDEX** incrementing by one until **TRIMCHEK** returns **LOOP=FALSE** (Input)
LOOP	A logical flag set by **TRIMCHEK** to indicate whether additional **MINDEX** subscripts are needed to complete the processing of all the Mach number/symmetry conditions on all the **TRIM** entries. One pass for each unique Mach number will be performed with **MINDEX** incrementing by one until **TRIMCHEK** returns **LOOP=FALSE** (Output)
GOAERO	A logical flag set by **TRIMCHEK** to indicate whether the **STEADY** module should be called to process the **MINDEX**'th Mach Number (Output)
CASE	A relation describing the Solution Control Case Definitions (Input)
TRIMDATA	An output relation that contains the description of the **TRIM** entries for each boundary condition and each subcase. The **TRIM** entries for each subcase are repeated, even if the data are the same so as to uniquely associate Mach Numbers, subcase identification numbers and **MINDEX** subscripts. Input accelerations are normalized to be in consistent units (Output)

Application Calling Sequence:

None

Method:

The **TRIMCHEK** preface module performs initial aerodynamic processing for planar **STEADY** aerodynamics. It is driven by the the **TRIM** data present in the bulk data packet and the **SAERO** disciplines in the **CASE** relation. The **CASE** relation provides the symmetries while the **TRIM** relation provides the Mach numbers. Only if **SAERO** disciplines are in **CASE** is any processing done and both **TRIM** and **AEROS** entries must be found.

On each call, the **PASSDF** submodule is called to determine the set of all Mach numbers and, for each Mach number, whether symmetric, antisymmetric or both boundary conditions are to be applied. Having determined all unique Mach numbers, the **PASSDF** then determines the **MINDEX**'th Mach number in numerical order (lowest to highest) and that is flagged to be processed using the **GOAERO** flag. If the chosen Mach number is the last one, the **LOOP** flag is set to false to tell the MAPOL sequence that no more calls are needed. At this point, the **GOAERO** flag will still be true to allow processing of the last Mach number. Only if no **SAERO** are called for in **CASE** will **GOAERO** be false on output.

Design Requirements:

1. The **TRIMCHEK** module interacts with the executive in that the **LOOP** variable is output on the first call and the module expects to be called again as long as **LOOP** is true. For each time called, the **MINDEX** parameter should be unique although it need not be monotonically increasing.

Error Conditions:

1. Errors in the **TRIM** specifications are flagged.

Engineering Application Module: **UNSTEADY**

Entry Point: **UNSTDY**

Purpose:

Unsteady aeroelastic analysis preface.

MAPOL Calling Sequence:

```
CALL UNSTEADY ( GEOMUA, AECOMPU, [AJJTL], [D1JK], [D2JK], [SKJ]
                AERUGEOM, CAROUGEO );
```

GEOMUA	A relation describing the aerodynamic boxes for the un**STEADY** aerodynamics model. The location of the box centroid, normal and pitch moment axis are given. It is used in splining the aerodynamics to the structure and to map responses back to the aerodynamic boxes (Output)
AECOMPU	A relation describing aerodynamic components for the un**STEADY** aerodynamics model. It is used in splining the aerodynamics to the structural model (Output)
[AJJTL]	A matrix containing the transposed unsteady **AIC** matrix for each Mach number, reduced frequency and symmetry option in the Bulk Data **MKAERO1** and **MKAERO2** entries (Output)
[D1JK]	Real part of the substantial derivative matrix (Output)
[D2JK]	Imaginary part of the substantial derivative matrix (Output)
[SJK]	Integration matrix to take pressures to force. (Output)
AERUGEOM	Relation containing the aerodynamic planform geometric grid points for the flutter model (Character, Output)
CAROUGEO	Relation containing the connectivity data for the aerodynamic planform geometric grid points for the flutter model (Character, Output)

Application Calling Sequence:

None

Method:

The unsteady aerodynamics preface module is activated under the following conditions:

1. If any **FLUTTER** cases are in the **CASE** relation
2. If any **BLAST** cases are in the **CASE** relation
3. If there are any **TRANSIENT** or **FREQUENCY** cases that invoke the **GUST** option.

After checking for the presence of the proper cases, the **MKAERO1** and **MKAERO2** data are read from the database and a list of all {symm,m,k} sets are assembled. The first record of the **UNMK** table is then written containing a count of the number of {m,k} pairs in each of the six symmetry classes. The second record will be loaded within the **APD** submodule and closed on return to **UNSTEADY.**

Then the unsteady aerodynamics model is read from the database into memory for the **APD** submodule. That module is then called to form the geometrical description of the unsteady model. The **ACPT**, the **GEOMUA** and the **AECOMPU** entities are written along with the second record of the **UNMK**.

Once the geometry data are complete, the **AMG** submodule is called to compute the **AJJTL**, **D1JK**, **D2JK** and **SKJ** matrix. These computations are done for all the {symm,m,k} sets in the bulk data. Each **AJJT** matrix is appended to the **AJJTL** output matrix. The **D1JK**, **D2JK** and **SJK** matrices will have two separate matrices stored in a similar fashion if and only if both subsonic and supersonic Mach numbers appear in the **UNMK** sets. Once these computations are complete, **UNSTEADY** returns control to the MAPOL sequence.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **WOBJGRAD**

Entry Point: **WOBJGD**

Purpose:

To compute weight function sensitivity to the design variables.

MAPOL Calling Sequence:

```
CALL WOBJGRAD ( NITER, NDV, GLBDES, DWGH1, DDWGH2 );
```

NITER	Design iteration number (Integer,Input)
NDV	Number of design variables (Integer,Input)
GLBDES	Relation of global design variables (Character,Input)
DWGH1	Unstructured entity of invariant linear portion of the weight function sensitivity (Character,Input)
DDWGH2	Unstructured entity of nonlinear portion of the weight function sensitivity (Character,Input)

Application Calling Sequence:

None

Method:

This module first reads all **DVID** from relation **GLBDES**. Then the module reads entities **DWGH1** and **DDWGH2** into memory. Because **DWGH1** and **DDWGH2** have the design variable **DVID** index as their first record, the weight function sensitivities may be generated by searching for this **DVID** in **DWGH1** and **DDWGH2** and obtaining the corresponding sensitivity terms.

Design Requirements:

None

Error Conditions:

None

Engineering Application Module: **YSMERGE**

Entry Point: **YSMERG**

Purpose:

To provide a special purpose merge utility for merging **YS**-like vectors (vectors of enforced displacements) into matrices for data recovery.

MAPOL Calling Sequence:

```
CALL YSMERGE ( [UN], [YS(BC)], [UF], [PNSF(BC)], DYNFLG );
```

[UN]	Matrix containing the nodal response quantities for the independent degrees of freedom (Output)
[YS(BC)]	Optional matrix containing the vector of enforced displacements on the single-point constraint degrees of freedom. If the **YS** argument is omitted, null vectors are merged (Input), where **BC** represents the MAPOL boundary condition loop index number
[UF]	The matrix of free nodal response quantities to be merged with the **YS** vector
[PNSF(BC)]	The partitioning vector splitting the independent degrees of freedom into the free and the single point constraint degrees of freedom (Input), where **BC** represents the MAPOL boundary condition loop index number
DYNFLG	Dynamic matrix form flag: if **DYNFLG** is nonzero, the matrix **UF** is assumed to have the form of a dynamic response matrix: three columns per subcase; (1) displacement, (2) velocity and (3) acceleration (Integer, Input)

Application Calling Sequence:

None

Method:

The **YSMERGE** engineering utility module is a general utility to merge a column vector, **YS**, (or a null column) that represents a partition of the desired output matrix with the other partition, **UF**, based on an input partitioning vector. The column dimension of **UF** is used to determine the number of times **YS** is to be duplicated in the merge operation. The result is loaded into the **UN** matrix. As a special option, the **DYNFLG** input is used to direct the module to assume that the **UF** and **UN** matrices have, or are to have, the form of a dynamic response "displacement" matrix. These matrices have three columns for each time/frequency step:

(1) Displacement

(2) Velocity

(3) Acceleration

When **DYNFLG** is nonzero, the **YS** matrix is merged with the first column (displacements) of each triplet with null partitions used for the corresponding velocities and accelerations.

Design Requirements:

1. The **YS** matrix entity, if it is included in the calling sequence, must be null (no columns) or be a column vector. If the matrix is null, the routine acts as though it were not included in the calling sequence.

Error Conditions:

None

THIS PAGE LEFT INTENTIONALLY BLANK

6. APPLICATION UTILITY MODULES

Large software systems such as ASTROS require that similar operations be performed in many code segments. To reduce the maintenance effort and to ease the programming task, a set of commonly used application utilities were identified and used whenever the application required those tasks to be performed. This section is devoted to the documentation of the set of application utilities in ASTROS. The suite of utilities in ASTROS includes small (performed entirely in memory) matrix operations like linear equation solvers, matrix multiplication and others. Another suite of utilities have been written to sort tables or columns of data on real, integer and character values in the table. Other utilities search lists of data stored in memory for particular key values, initialize arrays, operate on matrix entities and perform other disparate tasks of a general nature. The ASTROS user who intends to write application programs to be used within the ASTROS environment is strongly urged to study the suite of utilities documented in this section. ASTROS software designed to make use of the suite of application utilities can be much simpler to write, debug and maintain since these well-tested utilities can be substituted for code that would otherwise require programming effort.

The following subsections document the interface to the application utilities in two formats; using the executive system (MAPOL) and using the FORTRAN calling sequence. In most cases, there is no MAPOL language interface since these utilities are useful only within an application module. In other cases, however, the utility has been identified as a feature accessible through the executive. Finally, a small number of these application utilities are intended for access only by the executive system. This family of utilities is always associated with obtaining formatted output of data stored on the database.

Application Utility Module: **APPEND**

Entry Point: **APPEND**

Purpose:

This routine adds all the columns of one input matrix to the end of another.

MAPOL Calling Sequence:

```
CALL APPEND ( MATOUT, MATIN );
```

Application Calling Sequence:

```
CALL APPEND ( MATOUT, MATIN, IKOR )
```

MATOUT	The name of the output matrix to which the columns are added (Character, Output)
MATIN	The name of the input matrix from which the extra columns are extracted (Character, Input)
IKOR	Open core base address for local dynamic memory allocation (Integer, Input)

Method:

Matrix **MATOUT** is first initialized. Error checks are made to see if the matrices are conformable for the append operation. The columns of **MATIN** are then appended to the **MATOUT** matrix with special provisions given to handle null columns in **MATIN**.

Design Requirements:

None

Error Conditions:

1. **MATOUT** has not been created first (**APPEND** does an **MXINIT**, but the entity must already exist as a matrix).
2. **MATOUT** and **MATIN** have different types (precision/real or complex).
3. **MATOUT** and **MATIN** have a different number of rows.

Application Utility Module: **DAXB**

Entry Point: **DAXB**

Purpose:

This routine takes the double-precision cross product of vectors in a three-dimensional space.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DAXB ( A, B, C )
```

A	First vector (3x1) (Double, Input)
B	Second vector (3x1) (Double, Input)
C	Cross product **A X B** (Double, Output)

Method:

None

Design Requirements:

None

Error Conditions:

None

Application Utility Module: **GMMATC**

Entry Point: **GMMATC**

Purpose:

Perform the in-core complex matrix multiplications:

$$[A][B] = [C]$$
$$[A][B]^T = [C]$$
$$[A]^T[B] = [C]$$
$$[A]^T[B]^T = [C]$$

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL GMMATC ( A, IROWA, ICOLA, MTA, B, IROWB, ICOLB, NTB, C )
```

A	Matrix of **IROWA** rows and **ICOLA** columns stored in row order in a linear array (Character,Input)
B	Matrix of **IROWB** rows and **ICOLB** columns stored in row order in a linear array (Character,Input)
MTA,NTB	Transpose flags (Character,Input) = **0** if no transpose = **1** if transpose
C	Result of the matrix multiplication (Character,Output)

Method:

The **GMMATC** routine assumes that sufficient storage space is available in core to perform the multiplication. The matrices are checked to ensure that they are of proper dimensions to be multiplied. Complex single-precision is used throughout the routine.

Design Requirements:

None

Error Conditions:

None

Application Utility Module: **GMMATD**

Entry Point: **GMMATD**

Purpose:

Perform the in-core double-precision matrix multiplications:

$$[A][B] = [C]$$
$$[A][B]^T = [C]$$
$$[A]^T[B] = [C]$$
$$[A]^T[B]^T = [C]$$

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL GMMATD ( A, IROWA, ICOLA, MTA, B, IROWB, ICOLB, NTB, C )
```

A	Matrix of **IROWA** rows and **ICOLA** columns stored in row order in a linear array (Character,Input)
B	Matrix of **IROWB** rows and **ICOLB** columns stored in row order in a linear array (Character,Input)
MTA,NTB	Transpose flags (Integer,Input) = **0** if no transpose = **1** if transpose
C	Result of the matrix multiplication (Character,Output)

Method:

The **GMMATD** routine assumes that sufficient storage space is available in core to perform the multiplication. The matrices are checked to ensure that they are of proper dimensions to be multiplied. Double precision is used throughout the routine.

Design Requirements:

None

Error Conditions:

None

Application Utility Module: **GMMATS**

Entry Point: **GMMATS**

Purpose:

Perform the in-core single-precision matrix multiplications:

$$[A][B] = [C]$$
$$[A][B]^T = [C]$$
$$[A]^T[B] = [C]$$
$$[A]^T[B]^T = [C]$$

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL GMMATS ( A, IROWA, ICOLA, MTA, B, IROWB, ICOLB, NTB, C )
```

A	Matrix of **IROWA** rows and **ICOLA** columns stored in row order in a linear array (Character,Input)
B	Matrix of **IROWB** rows and **ICOLB** columns stored in row order in a linear array (Character,Input)
MTA,NTB	Transpose flags (Integer,Input) = **0** if no transpose = **1** if transpose
C	Result of the matrix multiplication (Character,Output)

Method:

The **GMMATS** routine assumes that sufficient storage space is available in core to perform the multiplication. The matrices are checked to ensure that they are of proper dimensions to be multiplied. Single precision is used throughout the routine.

Design Requirements:

None

Error Conditions:

None

Application Utility Module: **INVERC**

Entry Point: **INVERC**

Purpose:

Single precision complex in-core matrix inversion and linear equation solver. Finds solution to the matrix equation:

[A]{X} = {B}

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL INVERC ( NDIM, A, N, B, M, DETERM, ISING, WORK2 )
```

NDIM	The leading dimension of **A** as declared in the calling routine. **A(NDIM,N)** (Integer, Input).
A	Array containing the partition to be inverted. On output, the contents of the upper left **N** x **N** partition are replaced by the inverse. (Complex, Input)
N	Size of the upper left **A** partition to be inverted. (Integer, Input)
B	Column of constants (optional input of minimum size: **B(NDIM,1)**). On output, contains the solution vector(s) of the linear equations (Complex, Input)
M	Number of columns of **B** (Integer, Input)
DETERM	Determinant of **A** if nonsingular (Complex, Output)
ISING	Error flag = **1** if **A** nonsingular = **2** if **A** singular (Integer, Input and Output)
WORK2	Additonal working storage **(N,3)** (Complex, Input)

Method:

Note that all or the upper left square partition of the input array **A** may be inverted. If on input, the value of **ISING** is less than zero, the determinant of the **A** matrix is not calculated. The value of **DETERM** on return will be zero. The matrix inversion routine uses the Gauss-Jordian method with complete row-column interchange. Sufficient core storage must be set aside in **INDEX** to complete the inversion.

Error Conditions:

None

Application Utility Module: **INVERD**

Entry Point: **INVERD**

Purpose:

Double precision in-core matrix inversion and linear equation solver. Finds solution to the matrix equation:

[A]{X} = {B}

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL INVERD ( NDIM, A, N, B, M, DETERM, ISING, WORK2 )
```

NDIM	The leading dimension of **A** as declared in the calling routine. **A(NDIM,N)** (Integer, Input).
A	Array containing the partition to be inverted. On output, the contents of the upper left **N** x **N** partition are replaced by the inverse. (Double, Input)
N	Size of the upper left **A** partition to be inverted. (Integer, Input)
B	Column of constants (optional input of minimum size: **B(NDIM,1)**). On output, contains the solution vector(s) of the linear equations (Double, Input)
M	Number of columns of **B** (Integer, Input)
DETERM	Determinant of **A** if nonsingular (Double, Output)
ISING	Error flag = **1** if **A** nonsingular = **2** if **A** singular (Integer, Input and Output)
WORK2	Working storage **(N,3)** (Double, Input)

Method:

Note that all or the upper left square partition of the input array **A** may be inverted. If on input, the value of **ISING** is less than zero, the determinant of the A matrix is not calculated. The value of **DETERM** on return will be zero. The matrix inversion routine uses the Gauss-Jordian method with complete row-column interchange. Sufficient core storage must be set aside in **INDEX** to complete the inversion.

Error Conditions:

None

Application Utility Module: **INVERS**

Entry Point: **INVERS**

Purpose:

Single precision in-core matrix inversion and linear equation solver. Finds solution to the matrix equation:

[A]{X} = {B}

MAPOL Calling Sequence:

None

Application Calling Sequence:

CALL INVERS (NDIM, A, N, B, M, DETERM, ISING, WORK2)

NDIM	The leading dimension of **A** as declared in the calling routine. **A(NDIM,N)** (Integer, Input).
A	Array containing the partition to be inverted. On output, the contents of the upper left **N** x **N** partition are replaced by the inverse. (Real, Input)
N	Size of the upper left **A** partition to be inverted. (Integer, Input)
B	Column of constants (optional input of minimum size: **B(NDIM,1)**). On output, contains the solution vector(s) of the linear equations (Real, Input)
M	Number of columns of **B** (Integer, Input)
DETERM	Determinant of **A** if nonsingular (Real, Output)
ISING	Error flag = **1** if **A** nonsingular = **2** if **A** singular (Integer, Input and Output)
WORK2	Working storage **(N,3)** (Real, Input)

Method:

Note that all or the upper left square partition of the input array **A** may be inverted. If on input, the value of **ISING** is less than zero, the determinant of the A matrix is not calculated. The value of **DETERM** on return will be zero. The matrix inversion routine uses the Gauss-Jordian method with complete row-column interchange. Sufficient core storage must be set aside in **INDEX** to complete the inversion.

Error Conditions:

None

Application Utility Module: **MSGDMP**

Entry Point: **MSGDMP**

Purpose:

Retrieves messages queued by the **UTMWRT** module and writes them to the system output file.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MSGDMP
```

Method:

The **MSGDMP** routine reads the queued messages written by **UTMWRT** from the queue file and writes them onto the system output file. The queue file is then reset to accept the next set of messages. The intention is that **MSGDMP** will be called after each module's execution to allow easy determination of the last module executed, should the execution terminate.

Error Conditions:

None

Application Utility Module: **POLCOD**

Entry Point: **POLCOD**

Purpose:

This routine computes double-precision polynomial fit coefficients.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL POLCOD ( X, Y, N, S, COF )
```

X	The vector of scalar variables of length **N** (Input, Double Precision)
Y	The vector of terms such that **[Y(1)] = [Y]** at **X(1)** (Input, Double Precision)
N	The rank of vectors **X** and **Y** (Input, Integer)
S	The scratch array of length **N** to store the master polynomial coefficients (Input, Double Precision)
COF	The vector of coefficients such that **[Y(Z)] = [COF(1)] + Z[COF(2)] + Z**2[COF(3)] + ...** (Output, Double Precision)

Method:

This routine computes polynomial fit coefficients from solution of Vandermonde matrix equations. It is taken from "Numerical Recipes," Section 3.5, routine **POLCOE**.

Design Requirements:

1. Use **POLEVD** to evaluate the polynomial values based on the computed coefficients.
2. Use **POLSLD** to evaluate the polynomial derivative values based on the computed coefficients.

Error Conditions:

None

Application Utility Module: **POLCOS**

Entry Point: **POLCOS**

Purpose:

This routine computes single-precision polynomial fit coefficients.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL POLCOS ( X, Y, N, S, COF )
```

X	The vector of scalar variables of length **N** (Input, Real)
Y	The vector of terms such that **[Y(1)] = [Y]** at **X(1)** **[Y(2)] = [Y]** at **X(2)** ... (Input, Real)
N	The rank of vectors **X** and **Y** (Input, Integer)
S	The scratch array of length **N** to store the master polynomial coefficients (Input, Real)
COF	The vector of coefficients such that **[Y(Z)] = [COF(1)] + Z[COF(2)] + Z**2[COF(3)] + ...** (Output, Real)

Method:

This routine computes polynomial fit coefficients from solution of Vandermonde matrix equations. It is taken from "Numerical Recipes," Section 3.5, routine **POLCOE**.

Design Requirements:

1. Use **POLEVS** to evaluate the polynomial values based on the computed coefficients.
2. Use **POLSLS** to evaluate the polynomial derivative values based on the computed coefficients.

Error Conditions:

None

Application Utility Module: **POLEVD**

Entry Point: **POLEVD**

Purpose:

This routine performs double-precision polynomial evaluation from fit coefficients.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL POLEVD ( COF, N, X, Y )
```

COF	The vector of coefficients such that `[Y(Z)] = [COF(1)] + Z[COF(2)] + Z**2[COF(3)] + ...` (Input, Double Precision)
N	The rank of vectors **X** and **Y** (Input, Integer)
X	The scalar value at which polynomial is evaluated (Input, Double Precision)
Y	The function value at **X** (Output, Double Precision)

Method:

This routine performs double-precision polynomial evaluation from fit coefficients from solution of Vandermonde matrix equations. It is taken from "Numerical Recipes," Section 3.5, routine **POLCOE**.

Design Requirements:

1. Use **POLCOD** to evaluate the fit coefficients.
2. Use **POLSLD** to evaluate the polynomial derivative values based on the computed coefficients.

Error Conditions:

None

Application Utility Module: **POLEVS**

Entry Point: **POLEVS**

Purpose:

This routine performs single-precision polynomial evaluation from fit coefficients.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL POLEVS ( COF, N, X, Y )
```

COF	The vector of coefficients such that `[Y(Z)] = [COF(1)] + Z[COF(2)] + Z**2[COF(3)] + ...` (Input, Real)
N	The rank of vectors **X** and **Y** (Input, Integer)
X	The scalar value at which polynomial is evaluated (Input, Real)
Y	The function value at **X** (Output, Real)

Method:

This routine performs single-precision polynomial evaluation from fit coefficients from solution of Vandermonde matrix equations. It is taken from "Numerical Recipes," Section 3.5, routine **POLCOE**.

Design Requirements:

1. Use **POLCOS** to evaluate the fit coefficients.
2. Use **POLSLS** to evaluate the polynomial derivative values based on the computed coefficients.

Error Conditions:

None

Application Utility Module: **POLSLD**

Entry Point: **POLSLD**

Purpose:

This routine performs double-precision polynomial derivative evaluation from fit coefficients.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL POLSLD ( COF, N, X, Y )
```

COF	The vector of coefficients such that `[Y(Z)] = [COF(1)] + Z[COF(2)] + Z**2[COF(3)] + ...` (Input, Double Precision)
N	The rank of vectors **X** and **Y** (Input, Integer)
X	The scalar value at which polynomial is evaluated (Input, Double Precision)
Y	The slope of function at **X** (Output, Double Precision)

Method:

This routine performs double-precision polynomial derivative evaluation from fit coefficients from solution of Vandermonde matrix equations. It is taken from "Numerical Recipes," Section 3.5, routine **POLCOE**.

Design Requirements:

1. Use **POLCOD** to evaluate the fit coefficients.
2. Use **POLEVD** to evaluate the polynomial values based on the computed coefficients.

Error Conditions:

None

Application Utility Module: **POLSLS**

Entry Point: **POLSLS**

Purpose:

This routine performs single-precision polynomial derivative evaluation from fit coefficients.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL POLSLS ( COF, N, X, Y )
```

COF	The vector of coefficients such that `[Y(Z)] = [COF(1)] + Z[COF(2)] + Z**2[COF(3)] + ...` (Input, Real)
N	The rank of vectors **X** and **Y** (Input, Integer)
X	The scalar value at which polynomial is evaluated (Input, Real)
Y	The slope of function at **X** (Output, Real)

Method:

This routine performs single-precision polynomial derivative evaluation from fit coefficients from solution of Vandermonde matrix equations. It is taken from "Numerical Recipes," Section 3.5, routine **POLCOE**.

Design Requirements:

1. Use **POLCOD** to evaluate the fit coefficients.
2. Use **POLEVD** to evaluate the polynomial values based on the computed coefficients.

Error Conditions:

None

Application Utility Module: **PS**

Entry Point: **PS**

Purpose:

Character function returns the character string as the matrix precision needed for **MXINIT**, memory management and others based on the machine precision

MAPOL Calling Sequence:

None

Application Calling Sequence:

PS (TYPFLG)

TYPFLG	Character string, either "**R**" or "**C**" for real or complex (Character, Input)

Method:

PS returns character string **RDP** on double-precision machines or **RSP** on single-precision machines for input **TYPFLG** of **R**, in other words **PS('R')** returns either **RSP** or **RDP**. The complex equivalent **CDP** or **CSP** is returned if **TYPFLG** is **C**.

Design Requirements:

None

Error Conditions:

1. If **TYPFLG** in **PS** is neither "**R**" nor "**C**", **PS** will return blank.

Application Utility Module: **RDDMAT**

Entry Point: **RDDMAT**

Purpose:

Reads a double-precision matrix entity into memory.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL RDDMAT ( MATNAM, NROW, NCOL, BLKN, GRPN, PNTR, DKOR )
```

MATNAM	Input Matrix database entity (Character, Input)
NROW	The number of rows in the matrix (Integer, Output)
NCOL	The number of columns in the matrix (Integer, Output)
BLKN	The name of the open core block to which the data are written (Character, Input)
GRPN	The name of the open core group to which the data are written (Character, Input)
PNTR	The pointer relative to **DKOR** where the matrix data begin. (Integer, Output)
DKOR	The double-precision open core base address. (Double, Input)

Method:

The matrix is opened, its size determined and a memory block with group name **GRPN** and block name **BLKN** is allocated to hold the matrix data. The matrix is then read into core with special provisions being taken to handle the case of null columns. The matrix is then closed. The calling routine is responsible for freeing the memory block.

Design Requirements:

1. The matrix must be closed on calling this routine.

Error Conditions:

1. Insufficient open core memory will cause ASTROS termination.

Application Utility Module: **RDSMAT**

Entry Point: **RDSMAT**

Purpose:

Reads a single-precision matrix entity into memory.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL RDSMAT ( MATNAM, NROW, NCOL, BLKN, GRPN, PNTR, RKOR )
```

MATNAM	Input Matrix database entity (Character, Input)
NROW	The number of rows in the matrix (Integer, Output)
NCOL	The number of columns in the matrix (Integer, Output)
BLKN	The name of the open core block to which the data are written (Character, Input)
GRPN	The name of the open core group to which the data are written (Character, Input)
PNTR	The pointer realtive to **DKOR** where the matrix data begin. (Integer, Output)
RKOR	The single-precision open core base address. (Real, Input)

Method:

The matrix is opened, its size determined and a memory block with group name **GRPN** and block name **BLKN** is allocated to hold the matrix data. The matrix is then read into core with special provisions being taken to handle the case of null columns. The matrix is then closed. The calling routine is responsible for freeing the memory block.

Design Requirements:

1. The matrix must be closed on calling this routine.

Error Conditions:

1. Insufficient open core memory will cause ASTROS termination.

Application Utility Module: **SAXB**

Entry Point: **SAXB**

Purpose:

This routine takes the single-precision cross product of vectors in a three-dimensional space.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL SAXB ( A, B, C )
```

A	First vector (3x1) (Real, Input)
B	Second vector (3x1) (Real, Input)
C	Cross product **A X B** (Real, Output)

Method:

None

Design Requirements:

None

Error Conditions:

None

Application Utility Module: **SHAPEGEN**

Entry Point: **SHAPGN**

Purpose:

Generates a set of **SHAPE** Bulk Data entries for use in ASTROS based on the element centroids of the specified elements.

Mapol Calling Sequence:

```
CALL SHAPEGEN ( SHPGEN );
```

SHPGEN	Relation defining the design variable shape function identification number, the list of associated elements and the shape to be generated as the input from Bulk Data file (Character, Input)

Application Calling Sequence:

None

Method:

This module first read the input Bulk Data relation **SHPGEN**. Then for each relation **SHPGEN** entry, the design variable shape function identification number is set, the element summary data is obtained and the element centroid is computed for each associated element which is defined by **ELEMLIST**, finally the coefficients are computed and the **SHAPE** entry is written into the punch file.

Design Requirements:

None

Error Conditions:

None

Application Utility Module: **USETPRT**

Entry Point: **USETPR**

Purpose:

To print the structural set definition table for each boundary condition contained in the **USET** entity.

MAPOL Calling Sequence:

```
CALL USETPRT ( USET(BC), BGPDT(BC) )
```

USET	The **USET** entity for the current boundary condition (Character, Input)
BGPDT	The **BGPDT** entity for the current boundary condition (Character, Input)

Application Calling Sequence:

None

Method:

The **USET** entity is opened to determine which boundary condition is to be processed. The **CASE** relation is opened and the appropriate **TITLE**, **SUBTITLE** and **LABEL** information are obtained.

The **INTID**, **EXTID** and **FLAG** attributes of the **BGPDT** are brought into core and sorted on internal id. The **USET** record for the current boundary condition is also brought into an open core memory block. Each tuple of the **BGPDT** is processed; each point in the structural set has its corresponding **USET** bit mask decoded to determine to which structural sets the degree of freedom belongs. A running count in each dependent and independent structural set is maintained and echoed.

Error Conditions:

None

Application Utility Module: **UTCOPY**

Entry Point: **UTCOPY**

Purpose:

To copy a specified number of contiguous single-precision words from one location to another.

Application Calling Sequence:

```
CALL UTCOPY ( DEST, SOURCE, NWORD )
```

DEST	Source array to be copied (Character,Input)
SOURCE	Destination array (Character,Input)
NWORD	Number of single-precision words to be copied (Integer,Input)

Method:

The source and destination arrays are operated on as integer arrays inside the **UTCOPY** routine. If double-precision data are to be copied, the **NWORD** argument must be adjusted accordingly.

Design Requirements:

None

Error Conditions:

None

Application Utility Module: **UTCSRT**

Entry Point: **UTCSRT**

Purpose:

To sort a table of numbers on a four or eight character hollerith column of the table

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL UTCSRT ( ISORT, ITBROW, BOTLIM, TOPLIM, KEYPOS, TOTLEN, KEYLEN )
```

ISORT	Array to be sorted (Any, Input)
ITBROW	An array of length **TOTLEN** single-precision words used to store a table row (Any, Input)
BOTLIM	The location in the **ISORT** array of the first word of the first entry to be sorted (Integer, Input)
TOPLIM	The location in the **ISORT** array of the last word of the last entry to be sorted (Integer, Input)
KEYPOS	The column in the table of the first word of the 1 or two word character field on which the sort occurs. It must be a value between 1 and **TOTLEN** (Integer, Input)
TOTLEN	The length in single-precision words of one table row (Integer, Input)
KEYLEN	The number of characters in the hollerith string. Must be either four or eight. If it is not four, it is assumed to be eight without warning (Integer, Input)

Method:

The **UTCSRT** routine uses a QUICKSORT algorithm out lined in "The Art Of Computer Programming, Volume 3 / Sorting And Searching" by D.E. Knuth, Page 116. Several improvements have been made over the pure quicksort algorithm. The first is a random selection of the key value around which the array is sorted. This feature allows this routine to handle partially sorted information more rapidly than the pure quicksort algorithm. The second improvement in this routine is that a cutoff array length is used to direct further array sorting to an insert sort algorithm (Ibid. Page 81). This method has proven to be more rapid than allowing small arrays to be sorted by the quicksort algorithm. Presently this cutoff length is set at 15 entries. Studies should be conducted on each type of machine in order to set this cutoff length to maximize the speed of this routine. This sorting algorithm requires a integer stack in which to place link information during the sort. The maximum required size for this stack array in twice the natural log of the number of rows in the table. At present, the **UTCSRT** routine has hard coded an array of size **(2,40)** which provides for 1 trillion entries to be sorted.

Design Requirements:

None

Error Conditions:

None

Application Utility Module: **UTEXIT**

Entry Point: **UTEXIT**

Purpose:

To terminate the execution of the system when an error occurs.

MAPOL Calling Sequence:

```
CALL EXIT;
```

Application Calling Sequence:

```
CALL UTEXIT
```

Method:

The **UTEXIT** routine is called to cleanly terminate the execution of the ASTROS system. It calls the **DBTERM** database termination program to provide for normal closing of the database files, and dumps the queued messages from the **UTMWRT** utility. When these tasks have been completed, the program execution is terminated.

Design Requirements:

None

Error Conditions:

None

Application Utility Module: **UTGPRT**

Entry Point: **UTGPRT**

Purpose:

To print to the system output file the contents of special database matrix entities that have rows associated with structural degrees of freedom.

MAPOL Calling Sequence:

```
CALL UTGPRT ( BCID, USET(BCID), MAT1, MAT2, ..., MAT10 )
```

BCID	Boundary condition identification number (Integer,Input)
USET	Entity defining structural sets (Character,Input)
MATi	Matrix entity names (up to 10) (Character,Input)

Application Calling Sequence:

None

Method:

The matrix names are tested against the list of supported matrices. If the matrix entity matches one of the supported entities it is printed in a format based on the structural degrees of freedom (similar to displacement and eigenvector output from **OFP**). If the matrix name is not recognized, a call to **UTMPRT** is made instead to print the matrix out in standard banded format. The print format results in one line of output for each grid or scalar point in the structural model for each column of the matrix. Each line of output contains one value for each of the (up to) six degrees of freedom associated with it.

Design Requirements:

1. Only certain g-size matrices are printable in the format of this routine. The currently available matrices are: **DKUG**, **DMUG**, **DPVJ**, **DUG**, **DPGV**, **DUGV**, **DPTHVI**, **DPGRVI**, **PG**,and **DFDU**. Other matrices used in this routine will result in a call to the **UTMPRT** utility.

Error Conditions:

None

Application Utility Module: **UTMCOR**

Entry Point: **UTMCOR**

Purpose:

A special purpose utility to write an error message that insufficient open core is available in a functional module.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL UTMCOR ( MORCOR, TYPE, SUBNAM )
```

MORCOR	Integer containing the number of entries of type **TYPE** requested in the module terminating execution (Integer, Input)
TYPE	String identifying the type of data entries requested: = **RSP** for real, single-precision = **RDP** for real, double-precision = **CSP** for complex, single-precision = **CDP** for complex, double-precision = **CHAR** for character data (Character, Input)
SUBNAM	A character string containing the name of the module or subroutine that is terminating execution

Method:

The **UTMCOR** utility does an **MMSTAT** call to determine the maximum available open core. The **TYPE FLAG** is used to determine how many single-precision words are needed to satisfy the request for **MORCOR** entries. The difference between the required space and the maximum contiguous memory is used in a call to **UTMWRT** specifying the number of additional words needed. The **SUBNAM** is also sent to **UTMWRT** to identify the failure more precisely. Note that **TYPE=CHAR** is treated by **UTMCOR** as equivalent to **RSP**; the programmer must factor the number of words per entry and input **MORCOR** appropriately factored. After calling the message write utility, **UTMCOR** calls the **UTEXIT** utility to terminate the execution.

Design Requirements:

None

Error Conditions:

None

Application Utility Module: **UTMINT**

Entry Point: **UTMINT**

Purpose:

A special purpose utility to initialize a matrix entity of the machine precision to a diagonal or null matrix.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL UTMINT ( MATNAM, VAL, NROWS, NCOLS )
```

MATNAM	Character name of matrix entity to be initialized (Character, Input)
VAL	Single precision real value to initialize diagonal terms (Real, Input)
NROWS	The number of rows to be initialized (Integer, Input)
NCOLS	The number of columns to be initialized (Integer, Input)

Method:

The **UTMINT** uses the **DOUBLE** function to determine if the matrix to be initialized is single or double-precision. The requested entity is then opened and flushed. No check is made to ensure that the requested matrix exists. Based on the value of **VAL**, one of several paths through the utility is taken. If **VAL** is not zero, a diagonal matrix with diagonal terms given the value of **VAL** is created. If the non-zero value is 1.0 and **NROWS** equals **NCOLS**, the resulting identity matrix is specifically declared as such in the **MXINIT** call. If the matrix is rectangular, extra columns, if any, are null. If **VAL** is zero, a null matrix of the requested row and column dimensions is created. Note that all the matrices created by this utility are of the machine precision as determined by the **DOUBLE** function.

Design Requirements:

None

Error Conditions:

None

Application Utility Module: **UTMPRG,UTRPRG,UTUPRG**

Entry Point: **UTXPRG**

Purpose:

To purge the contents of database entities but leave the entity in existence.

MAPOL Calling Sequence:

```
CALL UTMPRG ( MAT1, MAT2, ..., MAT10 );
CALL UTRPRG ( REL1, REL2, ..., REL10 );
CALL UTUPRG ( UNS1, UNS2, ..., UNS10 );
```

Application Calling Sequence:

```
CALL DBFLSH ( ENTITY )
```

MATi	Matrix entity name (Character, Input)
RELi	Relation entity name (Character, Input)
UNSi	Unstructured entity name (Character, Input)
ENTITY	Any entity name (Character, Input)

Method:

The **UTMPRG**, **UTRPRG** and **UTUPRG** MAPOL calls are defined to allow up to 10 entities of a single type to be purged from the MAPOL sequence. The application interface is the **DBFLSH** routine which can take a single argument of an entity name of any type.

Design Requirements:

None

Error Conditions:

None

Application Utility Module: **UTMPRT**

Entry Point: **UTMPRT**

Purpose:

To print the contents of database matrix entities to the system output file.

MAPOL Calling Sequence:

```
CALL UTMPRT ( METHOD, MAT1, MAT2, ..., MAT10 );
```

Application Calling Sequence:

```
CALL UTMPRT ( MAT1, METHOD, IKOR, DKOR )
```

METHOD	Print method selection (optional for the MAPOL call) (Integer, Input)
MATi	Matrix entity name (Character, Input)
IKOR,DKOR	Base addresses of dynamic memory (Real and Double, Input)

Method:

If **METHOD** is zero (or absent from the MAPOL call), the matrix entity **MATi** is printed in a banded format: that is, all the terms from the first non-zero term to the last non-zero term (inclusive) are unpacked and printed. Null columns and groups of null columns are identified as such. Note that the MAPOL sequence call allows for up to 10 matrix entities to be printed. A nonzero **METHOD** prints the column by string with no intervening zeros.

Design Requirements:

None

Error Conditions:

None

Application Utility Module: **UTMWRT**

Entry Point: **UTMWRT**

Purpose:

This routine acts as the system message writer. It queues error messages to a temporary file for subsequent printing to the output file. The **MSGDMP** utility is used to actually print the queued messages.

MAPOL Calling Sequence:

None

Application Calling Sequence:

CALL UTMWRT (LEVEL, NUMBER, ARGMTS)

LEVEL	Severity level of the message (Integer,Input) **<** **0** No message header is written **=** **0** General Information **=** **1** System Fatal Message **=** **2** User Information Message **=** **3** User Warning Message **=** **4** User Fatal Message
NUMBER	Text string containing the message number in the form: **NN.MM.LL** (Character,Input)
ARGMTS	Text array containing arguments for the message text (Character,Input)

Method:

The **UTMWRT** routine cracks the message number **NUMBER** into its three component integers: **NN**, the module number, **MM**, the message number, and **LL**, the message length(in records). If **LL** is omitted (ie **NUMBER=NN.MM**), it defaults to one record in length.

The correct message text is then recovered from the message file by querying the **MSGLEN** for the module **NN** to obtain the starting record and adding the message number (**MM**) and message length (**LL**) to obtain the record numbers where the message text is stored. The message text is of the form:
'---text---$----text--$--.....'

If any **$** (dollar signs) exist in the message text, they are replaced by the **ARGMTS** supplied in the call statement. Note that the final message text including the **ARGMTS** must be less than 128 characters in length.

Design Requirements:

1. The pointers to the system database entity that contains the error message texts for each "module" must be stored in memory. Currently, the array for pointer storage is 200 words long which means that no more than 100 distinct "modules" can be defined. Note that this does not imply any limit on the number of error messages within a particular module's group of messages.

Error Conditions:

1. **UTMWRT** error: the number of modules exceeds the limit of **$**. This message results in program termination and can only be fixed by increasing the size of the message pointer storage array.

2. Error in **UTMWRT** when processing message number $. This message is a system level error which usually implies that a non-valid message number **NN.MM.LL** was passed to the module.

3. If the resultant message is longer than 128 characters, the unexpanded text is printed (with **$**'s) and the arguments are echoed.

Application Utility Module: **UTPAGE, UTPAG2**

Entry Points: **UTPAGE, UTPAG2**

Purpose:

To handle paging of the system output file during execution of the system. **UTPAG2** performs a page eject based on an anticipated number of lines to print.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL UTPAGE
CALL UTPAG2 ( N )
```

N	Number of lines that will be printed (Integer, Input)

Method:

The **UTPAGE** routine keeps track of the total line count and the line count for the current page. The total number of output lines allowed is maintained for use by this module. These quantities are stored in the **OUTPT1** common block. The **OUTPT2** common block is also used to store the header and titling data for the current execution. When output to the system output file is being performed, the line count is checked by the current module against the number of lines per page, when the maximum lines per page is reached, a call to **UTPAGE** causes a page advance on the system output file and the total number of printed lines is updated. The header information can be modified by the application modules by simply overwriting the current entries in the **OUTPT2** common block. Note that all system output should be performed using this utility module.

The **UTPAG2** routine performs a page eject if the **N** lines will not fit on the current page.

Design Requirements:

None

Error Conditions:

None

Application Utility Module: **UTRPRT**

Entry Point: **UTRPRT**

Purpose:

To print the contents of database relational entities to the system output file.

MAPOL Calling Sequence:

```
CALL UTRPRT ( REL1, REL2, ..., REL10 )
```

Application Calling Sequence:

```
CALL UTRPRT ( REL, IKOR, RKOR, DKOR )
```

REL	Relation to be printed (character,Input)
IKOR,RKOR, DKOR	Dynamic memory base addresses (Integer, Real and Double, Input)

Method:

The relational entity **RELi** is printed using the full relation projection. At present, if the full projection is too large to be output on one 132 character record, the remaining attributes are ignored. Each attribute, regardless of type, uses a 12 character format for output. The current version of **UTRPRT** has a few additional restrictions. The first is that any string attribute that is more than eight characters in length cannot be printed. The routine will ignore these attributes and write a message to that effect. In addition, double-precision attributes are first converted to single-precision before output.

Design Requirements:

1. Only the following attribute types are supported:
   INT, **KINT**, **AINT**
   RSP, **ARSP**
   RDP
   STR, **KSTR** (first 8 characters only)

Error Conditions:

1. Relational entity **REL** does not exist.
2. Relational entity **REL** is empty.
3. A string attribute cannot be printed when longer than eight characters.

Application Utility Module: **UTRSRT**

Entry Point: **UTRSRT**

Purpose:

To sort a table of numbers on a real column of the table

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL UTRSRT    ( ISORT, ITBROW, BOTLIM, TOPLIM, KEYPOS, TOTLEN )
```

ISORT	Array to be sorted (Any, Input)
ITBROW	An array of length **TOTLEN** single-precision words used to store a table row (Any, Input)
BOTLIM	The location in the **ISORT** array of the first word of the first entry to be sorted (Integer, Input)
TOPLIM	The location in the **ISORT** array of the last word of the last entry to be sorted (Integer, Input)
KEYPOS	The column in the table on which the sort occurs. It must be a value between 1 and **TOTLEN** (Integer, Input)
TOTLEN	The length in single-precision words of one table row (Integer, Input)

Method:

The **UTRSRT** routine uses a QUICKSORT algorithm outlined in "The Art Of Computer Programming, Volume 3 / Sorting And Searching" by D.E. Knuth, Page 116. Several improvements have been made over the pure quicksort algorithm. The first is a random selection of the key value around which the array is sorted. This feature allows this routine to handle partially sorted information more rapidly than the pure quicksort algorithm. The second improvement in this routine is that a cutoff array length is used to direct further array sorting to an insert sort algorithm (Ibid. Page 81). This method has proven to be more rapid than allowing small arrays to be sorted by the quicksort algorithm. Presently this cutoff length is set at 15 entries. Studies should be conducted on each type of machine in order to set this cutoff length to maximize the speed of this routine. This sorting algorithm requires a integer stack in which to place link information during the sort. The maximum required size for this stack array in twice the natural log of the number of rows in the table. At present, the **UTRSRT** routine has hard coded an array of size **(2,40)** which provides for 1 trillion entries to be sorted.

Design Requirements:

None

Error Conditions:

None

Application Utility Module: **UTSFLG, UTSFLR, UTGFLG, UTGFLR**

Entry Points: **UTSFLG, UTSFLR, UTGFLG, UTGFLR**

Purpose:

These routines set a named **FLAG** to an integer value, real value, or retrieve a value previously set.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL UTSFLG( INNAME , INVAL )
CALL UTSFLR ( INNAME , INVALR )
CALL UTGFLG( INNAME , OUTVAL )
CALL UTGFLR( INNAME , OUTVLR )
```

INNAME	The name of the **FLAG** to set (Character,Input)
INVAL	The value to set for the **FLAG** (Integer,Input)
INVALR	The value to set for the **FLAG** (Real,Input)
OUTVAL	The current value of the **FLAG** (Integer,Output)
OUTVLR	The current value of the **FLAG** (Real,Output)

Method:

None

Design Requirements:

None

Error Conditions:

None

Notes:

1. Routine **SETSYS** uses **UTSFLG** to set output file unit number, **PRINT** (set to second word of **/UNITS/** from **XXBD**) and to set the system precision **PREC** (=1 for single-precision; =2 for double-precision) based on the **DOUBLE** function. Large matrix utilities fetch these **FLAG** values by using **UTGFLG**.
2. Some of the **DEBUG** parameters are set by **UTSFLG** and are retrieved by the application modules using **UTGFLG**.
3. Routine **TIMCOM** uses **UTSFLR** to set system timing constants for matrix operations. The **FLAG**s are named: **TMUNIO**, **TMMXPT**, **TMMXUT**, **TMMXPK**, **TMMXUP**, **TMMXUM**, **TMMXPM**, **TMTRSP**, **TMTRDP**, **TMTCSP**, **TMTCDP**, **TMLRSP**, **TMLRDP**, **TMLCSP**, **TMLCDP**, **TMCRSP**, **TMCRDP**, **TMCCSP**, and **TMCCDP**. Large Matrix utilities fetch these constants by using **UTGFLR**.

Application Utility Module: **UTSORT**

Entry Point: **UTSORT**

Purpose:

To sort a table of data on an integer column of the table

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL UTSORT ( ISORT, ITBROW, BOTLIM, TOPLIM, KEYPOS, TOTLEN )
```

ISORT	Array to be sorted (Any, Input)
ITBROW	An array of length **TOTLEN** single-precision words used to store a table row (Any, Input)
BOTLIM	The location in the **ISORT** array of the first word of the first entry to be sorted (Integer, Input)
TOPLIM	The location in the **ISORT** array of the last word of the last entry to be sorted (Integer, Input)
KEYPOS	The column in the table on which the sort occurs. It must be a value between 1 and **TOTLEN**. (Integer, Input)
TOTLEN	The length in single-precision words of one table row (Integer, Input)

Method:

The **UTSORT** routine uses a QUICKSORT algorithm outlined in "The Art Of Computer Programming, Volume 3 / Sorting And Searching" by D.E. Knuth, Page 116. Several improvements have been made over the pure quicksort algorithm. The first is a random selection of the key value around which the array is sorted. This feature allows this routine to handle partially sorted information more rapidly than the pure quicksort algorithm. The second improvement in this routine is that a cutoff array length is used to direct further array sorting to an insert sort algorithm (Ibid. Page 81). This method has proven to be more rapid than allowing small arrays to be sorted by the quicksort algorithm. Presently this cutoff length is set at 15 entries. Studies should be conducted on each type of machine in order to set this cutoff length to maximize the speed of this routine. This sorting algorithm requires a integer stack in which to place link information during the sort. The maximum required size for this stack array in twice the natural log of the number of rows in the table. At present, the **UTSORT** routine has hard coded an array of size **(2,40)** which provides for 1 trillion entries to be sorted.

Design Requirements:

None

Error Conditions:

None

Application Utility Module: **UTSRCH**

Entry Point: **UTSRCH**

Purpose:

Search a table of values for an integer key from a table that is in sorted order on that integer key.

Application Calling Sequence:

```
CALL UTSRCH ( *ERR, KEY, LIST, LPNT, LSTLEN, INCR )
```

***ERR**	Error return if the **KEY** value is not found in the **LIST**.
KEY	Value being searched for in the **LIST** (Integer, Input)
LIST	Array in which the **KEY** should be located (Any, Input)
LPNT	On input, the pointer to the lowest key value in the **LIST** . On output, pointer to the matching value in the **LIST** (Integer)
LSTLEN	Length of the list including **INCR** - 1 trailing values following the last key (Integer, Input)
INCR	The spacing in the **LIST** between key values (Integer, Input)

Method:

The **UTSRCH** routine first calculates the number of key values to be searched. If there are less than a minimum number of key values (presently 15), then the list is searched sequentially. If more than the minimum exist, a binary search of the list is performed. If the value cannot be found, the routine returns to ***ERR**.

Design Requirements:

None

Error Conditions:

None

Application Utility Module: **UTSRT3**

Entry Point: **UTSRT3**

Purpose:

Sort a table on one to three integer keys.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL UTSRT3 ( Z, NENT, LENT, ZZ, KEY1, LKEY1, KEY2, LKEY2, KEY3, LKEY3, TYPE )
```

Z	Array to be sorted (Any, Input)
NENT	The number of rows (entries) in **Z**. (Integer, Input)
LENT	The number of words in each row of **Z** (Integer, Input)
ZZ	An array of length **LENT** to be used as intermediate storage (Integer, Input)
KEY1	Word offset in **Z** for the first key on which to sort. **KEY1** must be in the range 1 to **LENT** (Integer, Input)
LKEY1	Number of words in the first key on which to sort; use 0 if **KEY** is not used. **KEY1 + LKEY1** must be less than **LENT** (Integer, Input)
LKEY2	Number of words in the second key on which to sort; use 0 if **KEY** is not used. **KEY2 + LKEY2** must be less than **LENT** (Integer, Input)
LKEY3	Number of words in the third key on which to sort; use 0 if **KEY** is not used. **KEY3 + LKEY3** must be less than **LENT** (Integer, Input)
TYPE	Type of sort to perform (Integer, Input) ≥ **0** for sorting in increasing order < **0** for sorting in decreasing order

Method:

The **UTSRT3** routine sorts each key in order from one to three, with multiple-word keys being treated as though they were distinct integer keys on which the ascending or descending sort is performed.

Design Requirements:

1. There is an implementation limit of no more than 200 total keys.

```
LKEY1 + LKEY2 + LKEY3 < 201
```

Error Conditions:

1. Too many sort keys cause ASTROS termination.

Application Utility Module: **UTSRTD**

Entry Point: **UTSRTD**

Purpose:

Sort a vector of double-precision numbers.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL UTSRTD ( Z, BOTLIM, TOPLIM )
```

Z	Double precision array to be sorted (Double, Input)
BOTLIM	The location in the **Z** array of the first entry to be sorted (Integer, Input)
TOPLIM	The location in the **Z** array of the last entry to be sorted (Integer, Input)

Method:

The **UTSRTD** routine uses a QUICKSORT algorithm outlined in "The Art Of Computer Programming, Volume 3 / Sorting And Searching" by D.E. Knuth, Page 116. Several improvements have been made over the pure quicksort algorithm. The first is a random selection of the key value around which the array is sorted. This feature allows this routine to handle partially sorted information more rapidly than the pure quicksort algorithm. The second improvement in this routine is that a cutoff array length is used to direct further array sorting to an insert sort algorithm (Ibid. Page 81). This method has proven to be more rapid than allowing small arrays to be sorted by the quicksort algorithm. Presently this cutoff length is set at 15 entries. Studies should be conducted on each type of machine in order to set this cutoff length to maximize the speed of this routine. The algorithm used in this utility requires a stack array for storing the linking information generated during the sort. The maximum size needed for this stack is twice the natural log of the number of entries in the array. Currently, a stack of dimension (2,40) is hard coded which allows for a trillion entries to be sorted.

Design Requirements:

None

Error Conditions:

None

Application Utility Module: **UTSRTI**

Entry Point: **UTSRTI**

Purpose:

Sort a vector of integers.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL UTSRTI     ( Z, BOTLIM, TOPLIM )
```

Z	Integer array to be sorted (Integer, Input)
BOTLIM	The location in the **Z** array of the first entry to be sorted (Integer, Input)
TOPLIM	The location in the **Z** array of the last entry to be sorted (Integer, Input)

Method:

The **UTSRTI** routine uses a QUICKSORT algorithm outlined in "The Art Of Computer Programming, Volume 3 / Sorting And Searching" by D.E. Knuth, Page 116. Several improvements have been made over the pure quicksort algorithm. The first is a random selection of the key value around which the array is sorted. This feature allows this routine to handle partially sorted information more rapidly than the pure quicksort algorithm. The second improvement in this routine is that a cutoff array length is used to direct further array sorting to an insert sort algorithm (Ibid. Page 81). This method has proven to be more rapid than allowing small arrays to be sorted by the quicksort algorithm. Presently this cutoff length is set at 15 entries. Studies should be conducted on each type of machine in order to set this cutoff length to maximize the speed of this routine. The algorithm used in this utility requires a stack array for storing the linking information generated during the sort. The maximum size needed for this stack is twice the natural log of the number of entries in the array. Currently, a stack of dimension (2,40) is hard coded which allows for a trillion entries to be sorted.

Design Requirements:

None

Error Conditions:

None

Application Utility Module: **UTSRTR**

Entry Point: **UTSRTR**

Purpose:

Sort a vector of real numbers.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL UTSRTR     ( Z, BOTLIM, TOPLIM )
```

Z	Real array to be sorted (Real, Input)
BOTLIM	The location in the **Z** array of the first entry to be sorted (Integer, Input)
TOPLIM	The location in the **Z** array of the last entry to be sorted (Integer, Input)

Method:

The **UTSRTR** routine uses a QUICKSORT algorithm outlined in "The Art Of Computer Programming, Volume 3 / Sorting And Searching" by D.E. Knuth, Page 116. Several improvements have been made over the pure quicksort algorithm. The first is a random selection of the key value around which the array is sorted. This feature allows this routine to handle partially sorted information more rapidly than the pure quicksort algorithm. The second improvement in this routine is that a cutoff array length is used to direct further array sorting to an insert sort algorithm (Ibid. Page 81). This method has proven to be more rapid than allowing small arrays to be sorted by the quicksort algorithm. Presently this cutoff length is set at 15 entries. Studies should be conducted on each type of machine in order to set this cutoff length to maximize the speed of this routine. The algorithm used in this utility requires a stack array for storing the linking information generated during the sort. The maximum size needed for this stack is twice the natural log of the number of entries in the array. Currently, a stack of dimension (2,40) is hard coded which allows for a trillion entries to be sorted.

Design Requirements:

None

Error Conditions:

None

Application Utility Module: **UTSTOD, UTDTOS**

Entry Point: **UTSTOD, UTDTOS**

Purpose:

To convert a number of entries from single-precision to double-precision and copy them from one array into another and to convert a number of entries from double-precision to single-precision and copy them from one array into another.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL UTSTOD    ( RZ, DZ, TOTLEN )
CALL UTDTOS    ( DZ, RZ, TOTLEN )
```

RZ	Real array (Real,Input)
DZ	Double precision array (Double,Input)
TOTLEN	Length of array (Integer,Input)

Method:

For **UTSTOD, TOTLEN** entries of array **RZ** are copied to **DZ** and converted to double-precision. Similarly, for **UTDTOS**, the entries of **DZ** are copied to **RZ.**

Design Requirements:

None

Error Conditions:

None

Application Utility Module: **UTUPRT**

Entry Point: **UTUPRT**

Purpose:

To print the contents of database unstructured entities to the system output file.

MAPOL Calling Sequence:

```
CALL UTUPRT ( ENTNAM, TYPE )
```

ENTNAM	Entity name of array (Character,Input)
TYPE	Type of format to use in printing (Integer,Input) = **0** for Integer = **1** for Real = **2** for Double Precision

Application Calling Sequence:

None

Method:

The unstructured entity **ENTNAM** is printed to the system output file using the format specified by **TYPE**. The available formats for output are: **0** for Integer, **1** for Real, and **2** for Double Precision.

Design Requirements:

None

Error Conditions:

1. Unstructured entity **ENTNAM** does not exist.
2. Unstructured entity **ENTNAM** is empty.
3. Invalid unstructured type $ in **UTUPRT** print request for entity $. Valid types are: **0, 1, 2** (**INT**, **RSP**, **RDP**; respectively)

Application Utility Module: **UTZERD**

Entry Point: **UTZERD**

Purpose:

To initialize the contents of an array with a specified double-precision value.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL UTZERD ( ARRAY, NWORDS, VALUE )
```

ARRAY	Name of array (Double,Input and Output)
NWORDS	Length of array in words (Integer,Input)
VALUE	Value to use in initializing array (Double,Input)

Method:

NWORDS of array **ARRAY** are initialized with the value **VALUE**. Note that **VALUE** and **ARRAY** must be double-precision.

Design Requirements:

None

Error Conditions:

None

Application Utility Module: **UTZERS**

Entry Point: **UTZERS**

Purpose:

To initialize the contents of an array with a specified integer or real value.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL UTZERS ( ARRAY, NWORDS, VALUE )
```

ARRAY	Name of array (Any,Input and Output)
NWORDS	Length of array in words (Integer,Input)
VALUE	Value to use in initializing array (Integer or Real,Input)

Method:

NWORDS of array **ARRAY** are initialized with the value **VALUE**. Both **ARRAY** and **VALUE** must be single-precision

Design Requirements:

None

Error Conditions:

None

Application Utility Module: **XISTOI**

Entry Point: **XISTOI**

Purpose:

To convert a string to its integer equivalent.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL XISTOI ( STR, IVALUE, RC )
```

STR	Character string representing an integer number (Character,Input)
IVALUE	Resulting integer number (Integer,Output)
RC	Return Code (Integer,Output) = **0** if **STR** contained a legal integer = **1** if **STR** contained an illegal character = **2** if **STR** contained an illegal integer (overflow)

Method:

The character string **STR** is cracked one digit at a time with error checks made against the machine maximum integer to ensure that the resultant **IVALUE** is a legal integer.

Design Requirements:

1. Legal strings may contain plus (+), minus (-) and one or more decimal digits from 0 through 9.

Error Conditions:

1. Return codes

Application Utility Module: **XISTOR**

Entry Point: **XISTOR**

Purpose:

To convert a string to its real equivalent.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL XISTOR ( STR, VALUE, RC )
```

STR	Character string representing a real number (Character,Input)
VALUE	Resulting real number (Real,Output)
RC	Return Code (Integer,Output) = **0** if **STR** contained a legal real number = **1** if **STR** contained an illegal character = **2** if **STR** contained an overflow value = **3** if **STR** contained an underflow value

Method:

The character string **STR** is cracked into its three component parts: the leading whole number, the fractional digits and the exponential whole number. Each of the these pieces is optional. The three parts are then combined to form the real number.

Design Requirements:

1. Legal strings may contain plus (+), minus (-), one or more decimal digits from 0 through 9 and E or D and must contain a decimal point (.).
2. The Bulk Data style of real number representation is fully supported as is the FORTRAN **E**, **F** and **G** formats.

Error Conditions:

1. Return codes

7. LARGE MATRIX UTILITY MODULES

Finite element structural analysis, which forms the core of the ASTROS system, requires a suite of utilities for matrix operations which are able to efficiently handle very large, often sparse, matrices. This section is devoted to the documentation of the large matrix utilities in ASTROS. the designation ***large*** comes from the assumption made by each of these utilities that the relevant matrices are stored on the CADBB database and will be operated on in a fashion that allows them to be of arbitrary order. Other matrix operations are available in the general utility library documented in Section 6 for small matrices stored in memory. The suite of large matrix utilities in ASTROS includes partition/merge operations, decomposition and forward/backward substitutions, multiply/add and pure addition operations, transpose operations and real and complex eigenvalue extraction.

The following subsections document the interface to the large matrix utilities in two formats: using the executive system (MAPOL) and using the FORTRAN calling sequence. In some cases, the MAPOL language supports the particular matrix operation directly. In such cases, the user need not make a call to the particular utility, instead, the MAPOL compiler automatically generates the correct call to the appropriate utility. These direct interfaces are so indicated in the documentation.

Large Matrix Utility Module: **CDCOMP**

Entry Point: **CDCOMP**

Purpose:

To decompose a complex square matrix **[A]** into its upper and lower triangular factors:

[A] = [L] [U]

MAPOL Calling Sequence:

```
CALL DECOMP ([A], [L], [U]);
```

Note that the calling sequence for **CDCOMP** is through the MAPOL **DECOMP** module. The method is automatically selected if the input matrix is complex.

Application Calling Sequence:

```
CALL CDCOMP     (A, L, U, IKOR, RKOR, DKOR)
```

[A]	The matrix to be decomposed (Input, Character)
[L]	The lower triangular factor (Output, Character)
[U]	The upper triangular factor (Output, Character)
IKOR,RKOR,DKOR	The dynamic memory base address (Integer,Real and Double,Input)

Method:

The **CDCOMP** module decomposes complex matrices. The resultant lower, **[L]**, and upper, **[U]**, triangular factors are specially structured matrix entities having control information in the diagonal terms. They may only be reliably used by the back-substitution module **GFBS**.

Design Requirements:

1. The back-substitution phase of equation solving is performed with module **GFBS**.
2. The triangular factors **[L]** and **[U]** may not be used reliably by matrix utilities other than **GFBS**.

Error Conditions:

None

Large Matrix Utility Module: **CEIG**

Entry Point: **CEIG**

Purpose:

To solve the equation:

$$([M]p^2 + [B]p + [K])\{u\} = 0$$

for the eigenvalues **p** and the associated eigenvectors **{u}** where **[M]**, **[B]** and **[K]** are mass, damping and stiffness matrices, respectively.

MAPOL Calling Sequence:

```
CALL CEIG      ( SETID, BCID, USET, [KDD], [BDD], [MDD], LAMDAC, [CPHID],
                 [CPHIDL], NPHI );
```

Application Calling Sequence:

```
CALL CEIG      ( SETID, BCID, USET, KDD, BDD, MDD, LAMDAC, CPHID, CPHIDL,
                 OCEIGS, IKOR, RKOR, DKOR )
```

SETID	An optional set identification for the **EIGC** entry. Used to define the extraction parentheses if omitted from MAPOL or 0. The CASE **CMETHOD** attribute will define the **SETID** used.
BCID	The boundary condition identification number (Integer, Input)
USET	Entity defining structural sets for the current BC
[KDD]	Dynamic stiffness matrix - D-set (Input, Character)
[BDD]	Dynamic damping matrix - D-set (Input, Character)
[MDD]	Dynamic mass matrix - D-set (Input, Character)
LAMDAC	A relation entity containing a list of extracted complex eigenvalues (Output, Character)
[CPHID]	A matrix whose columns are the complex eigenvectors corresponding to the extracted eigenvalues (Output, Character)
[CPHIDL]	A matrix containing the left complex eigenvectors (Output, Character)
NPHI	The number of complex eigenvectors computed (Output, Integer)
OCEIGS	The name of the output entity for statistical information (Character)
IKOR,RKOR,DKOR	The dynamic memory base address (Integer,Real and Double,Input)

Method:

The Complex Eigenvalue Analysis Module calculates the eigenvalues and eigenvectors for a general system which may have complex terms in the mass, damping and stiffness matrices. The eigenvectors are scaled according to the user requested normalization scheme (**MAX** or **POINT**).The eigenvalues p and the eigenvectors **{u}** are always treated as complex. These data are related to the ud displacements if a direct formulation is used or are related to the uh displacements if a modal formulation is used.

Presently, the complex eigenvalue analysis is used by manually inserting a call to module **CEIG** in the MAPOL sequence. The relation **EIGC** will be automatically retrieved in module **CEIG** and the first

method that appears in the relation will control the extraction. The Inverse Power Method or the Upper Hessenburg Method which is selected by **EIGC** data is used to solve the eigenvalue problem. (Subroutines **CINVPR** or **HESS1**). In case there is insufficient core for Upper Hessenburg Method, the Inverse Power Method will be used if the necessary data exist on **EIGC**.

Design Requirements:

1. The matrices **[KDD]**, **[BDD]** and **[MDD]** must be complex, and matrices **[BDD]** and **[CPHIDL]** are not required.

Error Conditions:

1. **EIGC** is not in the Bulk Data packet.
2. **[KDD]** and/or **[MDD]** do not exist.
3. **[KDD]** and **[MDD]** are not compatible.
4. **[MDD]** is singular in **HESS** method.

Large Matrix Utility Module: **COLMERGE**

Entry Point: **MXMERG**

Purpose:

To merge two submatrices into a single matrix **[A]** column-wise:

$$[A] \leftarrow [A_{11} \quad A_{12}]$$

MAPOL Calling Sequence:

```
CALL COLMERGE  ([A], [A11], [A12], [CP]);
```

Application Calling Sequence:

```
CALL MXMERG    (A, A11, BLANK, A12, BLANK, CP, BLANK, KORE)
```

[A]	The resulting merged matrix (Output, Character)
[Aij]	The input partitions as shown above (Input, Character)
[CP]	The column partitioning vector (Input, Character)
BLANK	A character blank (Input, Character)
KORE	The dynamic memory base address (Integer,Input)

Method:

The partitioning vector **[CP]** must be a column vector containing zero and nonzero terms. The **[A11]** partition will be placed in **[A]** at positions where **[CP]** is zero, and the **[A12]** partition is placed in **[A]** at positions where **[CP]** is nonzero. If either of the partitions **[A11] or [A12]** is null, it may be omitted from the MAPOL calling sequence or a **BLANK** may be used in the application calling sequence.

The **COLPART** large matrix utility module performs the inverse of this module.

Design Requirements:

None

Error Conditions:

None

Large Matrix Utility Module: **COLPART**

Entry Point: **MXPRTN**

Purpose:

To partition a matrix **[A]** into two submatrices column-wise:

$$[A] \rightarrow [A_{11} \quad A_{12}]$$

MAPOL Calling Sequence:

```
CALL COLPART   ([A], [A11], [A12], [CP]);
```

Application Calling Sequence:

```
CALL MXPRTN    (A, A11, BLANK, A12, BLANK, CP, BLANK, KORE)
```

[A]	The matrix being partitioned (Input, Character)
[Aij]	The resulting partitions shown above (Output, Character)
[CP]	The column partitioning vector (Input, Character)
BLANK	A character blank (Input, Character)
KORE	The dynamic memory base address (Integer,Input)

Method:

The partitioning vector **[CP]** must be a column vector containing zero and nonzero terms. The **[A11]** partition will then contain those columns of **[A]** corresponding to a zero value in **[CP]**, and the **[A12]** partition is placed in **[A]** at positions where **[CP]** is nonzero. If either partition is not desired, it may be omitted from the MAPOL calling sequence or a **BLANK** may be used in the application calling sequence.

Design Requirements:

None

Error Conditions:

None

Large Matrix Utility Module: **DECOMP**

Entry Point: **DECOMP**

Purpose:

To decompose a general square matrix **[A]** into its upper and lower triangular factors:

[A] = [L][U]

MAPOL Calling Sequence:

```
CALL DECOMP     ([A], [L], [U]);
```

Application Calling Sequence:

```
CALL DECOMP     (A, L, U, IKOR, RKOR, DKOR)
```

[A]	The matrix to be decomposed (Input, Character)
[L]	The lower triangular factor (Output, Character)
[U]	The upper triangular factor (Output, Character)
IKOR,RKOR,DKOR	The dynamic memory base address (Integer, Real and Double,Input)

Method:

The **DECOMP** module can decompose both real and complex matrices. The resultant lower, **[L]**, and upper, **[U]**, triangular factors are specially structured matrix entities having control information in the diagonal terms. They may only be reliably used by the back-substitution module **GFBS**.

Design Requirements:

1. **DECOMP** can process both real and complex machine-precision matrices.
2. The back-substitution phase of equation solving is performed with module **GFBS**.
3. The triangular factors **[L]** and **[U]** may not be used reliably by matrix utilities other than **GFBS**.

Error Conditions:

None

Large Matrix Utility Module: **FBS**

Entry Point: **FBS**

Purpose:

To perform the forward/backward substitution phase of equation solving for symmetric matrices that have been decomposed with module **SDCOMP**.

MAPOL Calling Sequence:

```
CALL FBS        ([L], [RHS], [ANS], ISIGN);
```

Application Calling Sequence:

```
CALL FBSS       (L, RHS, ANS, ISIGN, IKOR, RKOR, DKOR)
```

[L]	The lower triangular decomposition factor obtained from **SDCOMP** (Input, Character)
[RHS]	The matrix of right-hand sides of the equations being solved (Input, Character)
[ANS]	The matrix of resulting solutions of the equations (Output, Character)
ISIGN	Sign of the right-hand sides in **[RHS]** (+1 for positive, -1 for negative) (Input, Integer)
IKOR,RKOR,DKOR	The dynamic memory base address (Integer, Real and Double,Input)

Method:

Given a real symmetric system of equations

$$[K][X] = \pm[P]$$

the **SDCOMP** large matrix utility is used to compute

$$[K] = [L][D][L]^T$$

such that **[D]** is a diagonal matrix. This module then completes the solution for **[X]** as

$$[L][Y] = \pm[P]$$

$$[L]^T[X] = [D]^{-1}[Y]$$

If **[RHS]** is blank, the inverse of the decomposed matrix will be returned in **[ANS]**.

Design Requirements:

None

Error Conditions:

None

Large Matrix Utility Module: **GFBS**

Entry Point: **GFBS**

Purpose:

To perform the forward/backward substitution phase of equation solving for general matrices that have been decomposed with module **DECOMP**.

MAPOL Calling Sequence:

```
CALL GFBS      ([L], [U], [RHS], [ANS], ISIGN);
```

Application Calling Sequence:

```
CALL GFBS      (L, U, RHS, ANS, ISIGN, IKOR, RKOR, DKOR)
```

[L], [U]	The names of the lower and upper triangular decomposition factors from **DECOMP** (Input, Character)
[RHS]	The matrix of right-hand sides of the equation being solved (Input, Character)
[ANS]	The matrix of resulting solutions of the equations (Output, Character)
ISIGN	Sign of the right-hand sides in **[ANS]** (Input,+1 for positive, -1 for negative) (Input, Integer)
IKOR,RKOR,DKOR	The dynamic memory base address (Integer, Real and Double,Input)

Method:

Given a general, real, or complex system of equations

$$[K][X] = \pm[P]$$

the **DECOMP** large matrix utility is used to compute

$$[K] = [L][U]$$

This module then completes the solution for **[X]** as:

$$[L][Y] = \pm[P]$$

$$[U][X] = [Y]$$

If **[RHS]** is blank, the inverse of the decomposed matrix will be returned in **[ANS]**.

Design Requirements:

None

Error Conditions:

None

Large Matrix Utility Module: **MERGE**

Entry Point: **MXMERG**

Purpose:

To merge four submatrices into a single matrix **[A]** based on one or two partitioning vectors.

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \rightarrow [A]$$

MAPOL Calling Sequence:

```
CALL MERGE     ([A], [A11], [A21], [A12], [A22], [CP], [RP]);
```

Application Calling Sequence:

```
CALL MXMERG    (A, A11, A21, A12, A22, CP, RP, KORE)
```

[A]	The resulting merged matrix (Output, Character)
[Aij]	The input partitions as shown above (Input, Character)
[RP]	The row partitioning vector (Input, Character)
[CP]	The column partitioning vector (Input, Character)
KORE	The dynamic memory base address (Integer,Input)

Method:

The partitioning vectors **[CP]** and **[RP]** must be column vectors containing zero and nonzero terms. The **[A11]** partition will be placed in **[A]** at positions where both **[RP]** and **[CP]** are zero. The **[A12]** partition will be placed in **[A]** at positions where **[RP]** is zero and **[CP]** is nonzero. The other partitions are treated in a similar manner.

If some of the partitions are null, they may be omitted from the MAPOL calling sequence or a character blank may be used in the application calling sequence. In a similar manner, if the row and column partition vectors are the same, one of them may be omitted or left blank in the MAPOL call. They must both be present in the application call.

If a row or column merge alone is required in the MAPOL sequence, the special purpose MAPOL utilities **ROWMERGE** and **COLMERGE** may be used.

Design Requirements:

None

Error Conditions:

None

Large Matrix Utility Module: **MPYAD**

Entry Point: **MPYAD**

Purpose:

To perform the general matrix multiply and add operations as shown below:

(1) [D] = ±[A][B]±[C] or = ±[A][B]

(2) [D] = ±[A]^{T}[B]±[C] or = ±[A]^{T}[B]

MAPOL Calling Sequence:

None, the MAPOL syntax supports algebraic matrix operations directly

```
[D] := ±[A]*[B] ±[C];

[D] := ±TRANS([A])*[B]±[C];
```

Application Calling Sequence:

CALL MPYAD (A, B, C, D, TFLAG, SIGNAB, SIGNC, IKOR, RKOR, DKOR)

A	The name of the input **A** matrix (Character)
B	The name of the input **B** matrix (Character)
C	The name of the input **C** matrix or blank (Character)
D	The name of the output **D** matrix (Character)
TFLAG	The transpose flag = 0 no transpose = 1 transpose matrix **A** (Integer, Input)
SIGNAB	The sign on the [A] [B] product = +1 +[A][B] = -1 -[A][B] (Integer, Input)
SIGNC	The sign on the [C] matrix = +1 +[C] = 0 no [C] matrix = -1 -[C] (Integer, Input)
IKOR,RKOR, DKOR	The dynamic memory base address (Integer, Real and Double,Input)

Method:

If no **[C]** matrix exists, the **C** argument should be blank and the **SIGNC** argument should be zero.

Design Requirements:

None

Error Conditions:

None

Large Matrix Utility Module: **MXADD**

Entry Point: **MXADD**

Purpose:

To perform the general matrix addition as shown below:

$$[C] = \alpha[A] + \beta[B]$$

MAPOL Calling Sequence:

None, the MAPOL syntax supports algebraic matrix operations directly.

$$[C] := (\alpha)\ [A] \pm (\beta)\ [B]$$

Application Calling Sequence:

CALL MXADD (A, B, C, ALPHA, BETA, DKOR, IKOR)

A	The name of the input **A** matrix (Character)
B	The name of the input **B** matrix (Character)
C	The name of the output **C** matrix (Character)
ALPHA	The constant complex multiplier of matrix **A**. Real array of length 2, the first word is the real part of the constant, the second is the imaginary part. (Input,Complex)
BETA	As **ALPHA** for the **B** matrix. (Input,Complex)
DKOR, IKOR	The dynamic memory base address (Double and Integer,Input)

Method:

None

Design Requirements:

None

Error Conditions:

None

Large Matrix Utility Module: **PARTN**

Entry Point: **MXPRTN**

Purpose:

To partition a matrix **[A]** into four submatrices based on one or two partitioning vectors:

$$[A] \rightarrow \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

MAPOL Calling Sequence:

```
CALL PARTN     ([A], [A11], [A21], [A12], [A22], [CP], [RP]);
```

Application Calling Sequence:

```
CALL MXPRTN    (A, A11, A21, A12, A22, CP,  RP,  KORE)
```

[A]	The matrix to be partitioned (Input, Character)
[Aij]	The resulting partitions as shown above (Output, Character)
[RP]	The row partitioning vector (Input, Character)
[CP]	The column partitioning vector (Input, Character)
KORE	The dynamic memory base address (Integer,Input)

Method:

The partitioning vectors **[CP]** and **[RP]** must be column vectors containing zero and nonzero terms. The **[A11]** partition will be formed from **[A]** at positions where both **[RP]** and **[CP]** are zero. The **[A12]** partition will be formed from **[A]** at positions where **[RP]** is zero and **[CP]** is nonzero. The other partitions are treated in a similar manner.

If some of the partitions are not desired as output, they may be omitted from the MAPOL calling sequence or a character blank may be used in the application calling sequence. In a similar manner, if the row and column partition vectors are the same, one of them may be omitted or left blank in the MAPOL call. They must both be present in the application call.

If a simple row or column partition is required in the MAPOL sequence, the special purpose MAPOL utilities **ROWPART** and **COLPART** may be used.

Design Requirements:

None

Error Conditions:

None

Large Matrix Utility Module: **REIG**

Entry Point: **REIG**

Purpose:

To solve the equation:

$$[\ [K] - \lambda[M]\ ]\{u\} = 0$$

for its eigenvalues, λ, and their associated eigenvectors $\{\varphi\}$.

MAPOL Calling Sequence:

```
CALL REIG       (ITER, BCID, USET(BCID), [K], [M], [MR], [DM], LAMA,
                 [PHI], [MI], NPHI);
```

Application Calling Sequence:

```
CALL REIG       (ITER, BCID, USET, K, M, MR, DM, LAMA, PHI, MI,
                 OEIGS, RKOR, DKOR)
```

ITER	The design iteration number (Integer, Input)
BCID	The boundary condition identification number (Integer, Input)
USET	The entity defining structural sets for the current boundary condition
[K], [M]	The stiffness and mass matrices (Input, Character)
[MR]	The rigid body mass matrix (Input, Character)
[DM]	The rigid body transformation matrix (Input, Character)
LAMA	Relation containing a list of extracted eigenvalues (Output, Character)
[PHI]	A matrix whose columns are the eigenvectors corresponding to the extracted eigenvalues (Output, Character)
[MI]	The modal mass matrix (Output, Character)
NPHI	The number of eigenvectors computed (Integer, Output)
OEIGS	The name of the output entity for statistical information (Character)
RKOR,DKOR	The dynamic memory base address (Real and Double,Input)

Method:

The matrices **[K]** and **[M]** must be real and the rigid body mass matrix **[MR]** and the rigid body transformation matrix **[DM]** are not required. The **REIG** module must query the **CASE** relational entity to determine which set of **EIGR** eigenvalue extraction data to use. Because of the multidisciplinary nature of the code, **REIG** assumes that, if called, an eigenanalysis is required. It uses the **EIGR** data that correspond to the selection for the current boundary condition, **BCID**.

Design Requirements:

None

Error Conditions:

None

Large Matrix Utility Module: **ROWMERGE**

Entry Point: **MXMERG**

Purpose:

To merge two submatrices into a single matrix **[A]** row-wise:

$$[A] \leftarrow \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}$$

MAPOL Calling Sequence:

```
CALL ROWMERGE ([A], [A11], [A21], [RP]);
```

Application Calling Sequence:

```
CALL MXMERG (A, A11, A21, BLANK, BLANK, BLANK, RP, KORE)
```

[A]	The resulting merged matrix (Output, Character)
[A11], [A21]	The input partitions as shown above (Input, Character)
[RP]	The row partitioning vector (Input, Character)
BLANK	A character blank (Input, Character)
KORE	The dynamic memory base address (Integer,Input)

Method:

The partitioning vector **[RP]** must be a column vector containing zero and nonzero terms. The **[A11]** partition will be placed in **[A]** at positions where **[RP]** is zero. If either of the partitions **[A11]** or **[A12]** is null, it may be omitted from the MAPOL calling sequence or a **BLANK** may be used in the application calling sequence.

The **ROWPART** large matrix utility module performs the inverse of this module.

Design Requirements:

None

Error Conditions:

None

Large Matrix Utility Module: **ROWPART**

Entry Point: **MXPRTN**

Purpose:

To partition a matrix **[A]** into two submatrices row-wise:

$$[A] \rightarrow \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}$$

MAPOL Calling Sequence:

```
CALL ROWPART ([A], [A11], [A21], [RP]);
```

Application Calling Sequence:

```
CALL MXPRTN (A, A11, A21, BLANK, BLANK, BLANK, RP, KORE)
```

[A]	The matrix being partitioned (Input, Character)
[Aij]	The resulting partitions shown above (Output, Character)
[RP]	The partitioning vector (Input, Character)
BLANK	A character blank (Input, Character)
KORE	The dynamic memory base address (Integer,Input)

Method:

The partitioning vector **[RP]** must be a column vector containing zero and nonzero terms. The **[A11]** partition will then contain those columns of **[A]** corresponding to a zero value in **[RP]**. If either partition is not desired as output, it may be omitted from the MAPOL calling sequence or a **BLANK** may be used in the application calling sequence.

Design Requirements:

None

Error Conditions:

None

Large Matrix Utility Module: **SDCOMP**

Entry Point: **SDCOMP**

Purpose:

To decompose a symmetric square matrix **[A]** into the form:

$$[A] \rightarrow [L][D][L]^T$$

where **[L]** is a lower triangular factor and the diagonal matrix and **[D]** has been stored on the diagonal of **[L]**.

MAPOL Calling Sequence:

```
CALL SDCOMP ([A], [L], USET(BC), SETNAM);
```

Application Calling Sequence:

```
CALL SDCOMP (A, L, CHLSKY, USET, SETNAM, IKOR, RKOR, DKOR)
```

[A]	The matrix to be decomposed (Input)
[L]	The lower triangular factor (Output)
CHLSKY	The input selection of Cholesky decomposition = **0** no Cholesky = **1** use Cholesky (Integer)
USET	The entity defining structural sets for the current boundary condition
SETNAM	The current structural set name
IKOR,RKOR,DKOR	The dynamic memory base address (Integer,Real and Double,Input)

Method:

The **SDCOMP** module can decompose real and complex symmetric matrices. The resultant lower factor, **[L]**, is a specially structured matrix entity having the terms of **[D]** on the diagonals. It may, therefore, only be reliably used by the back-substitution module, **FBS**.

Design Requirements:

None

Error Conditions:

1. Matrix **A** is singular.

Large Matrix Utility Module: **TRNSPOSE**

Entry Point: **TRNSPZ**

Purpose:

To generate the transpose of a matrix.

$$[A] \rightarrow [A]^T$$

MAPOL Calling Sequence:

```
CALL TRNSPOSE ([A], [ATRANS]);
```

Application Calling Sequence:

```
CALL TRNSPZ (A, ATRANS, IKOR, DKOR)
```

[A]	The name of the input matrix to be transposed (Input, Character)
[ATRANS]	The name of the resulting transposed matrix (Output, Character)
IKOR,DKOR	The dynamic memory base address (Integer and Double,Input)

Method:

The output matrix entity, **[ATRANS]**, must already exist on the database. It will be flushed and loaded by the transpose utility. All matrix types and precisions are supported. As a special feature, the user controlled 11th through 20th words of the **INFO** array for the input matrix are copied onto the transposed matrix.

Design Requirements:

1. The spill logic for the utility has a limit of eight scratch files to perform the transpose. If the transpose cannot be performed in eight passes using the available memory, the utility will terminate.

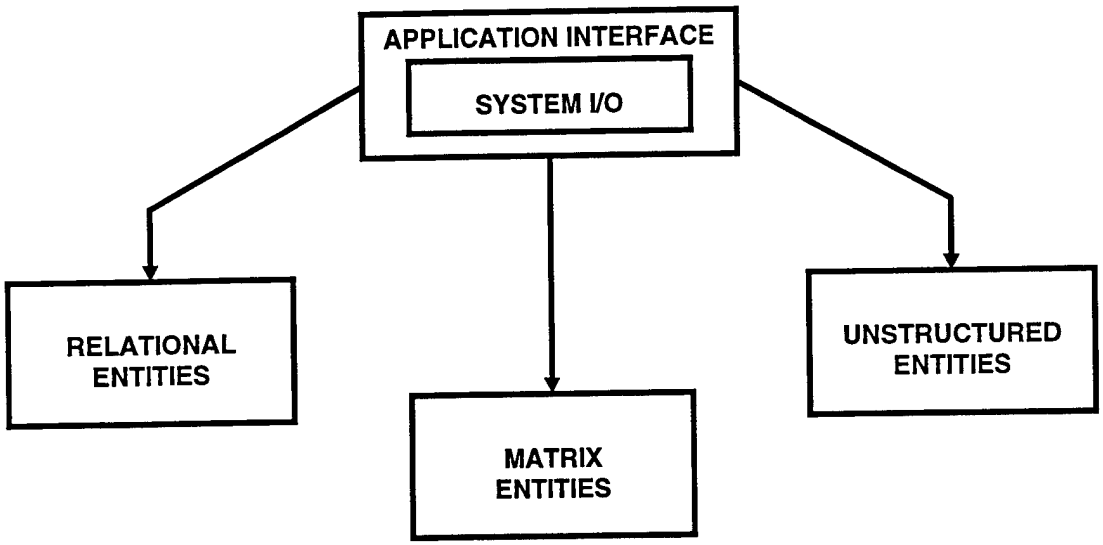
Error Conditions:

None

THIS PAGE LEFT INTENTIONALLY BLANK

8. THE CADDB APPLICATION INTERFACE

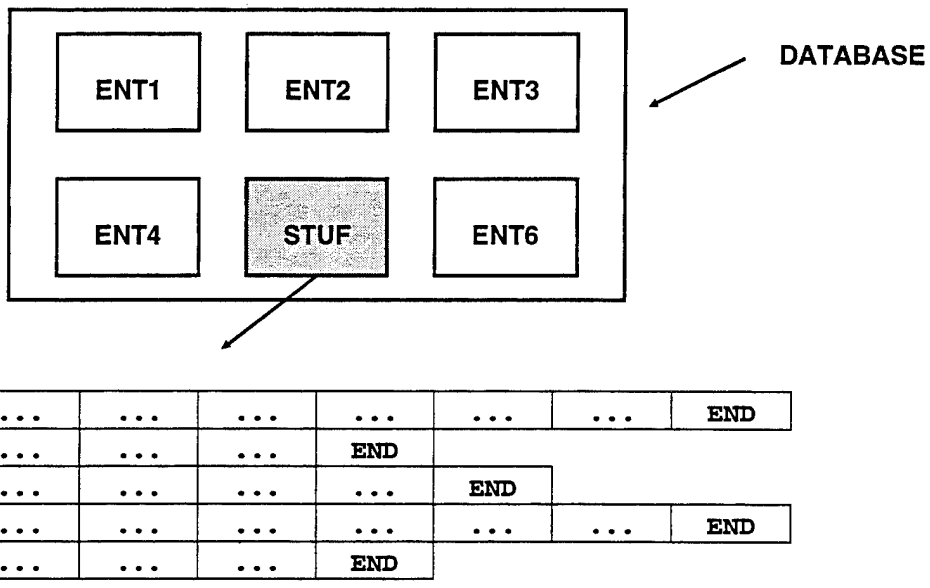
The Computer Automated Design Database (CADDB) is the heart of the ASTROS software system. It has been designed to provide the structures and access features typically required for scientific software applications development. CADDB can be viewed as a set of data entities that are accessible by a suite of utility routines called the application interface as shown below:



There are three types of entities: ***Unstructured***, ***Relational***, and ***Matrix***. These are described individually in the following paragraphs.

Unstructured Entities.

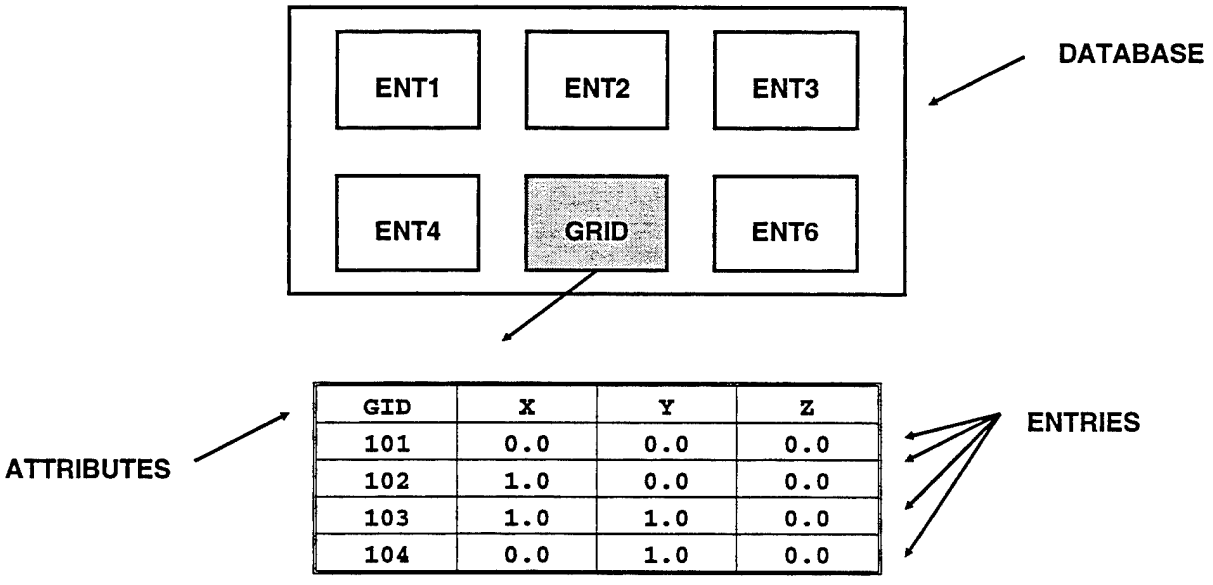
Unstructured entities form the least organized data type that may be used. An unstructured entity may be considered as a set of variable length records which have no predetermined structure and which may or may not have any relationship with each other. This is illustrated by the following:



Unstructured entities are typically used when "scratch" space is needed in an essentially sequential manner. Two important points, however, are that each record may be accessed randomly if the entity is created with an index structure, and that records may be read or written either in their entirety or only partially. Details of these features are discussed in Subsection 8.6.

Relational Entities.

Relational entities are very highly structured tables of data. The rows of the table are called ***entries*** or ***tuples*** and the columns are called ***attributes***, as shown below:



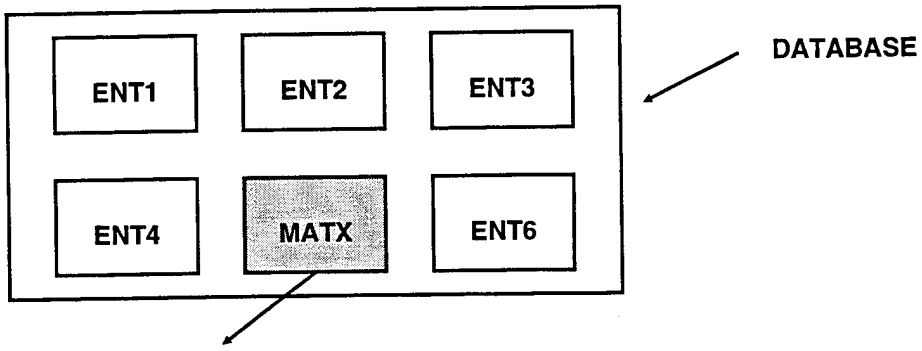
The definition of the attributes and their types is called the ***schema*** of the relation. Because the schema is an inherent part of a relation's data structure, each attribute may be referred to by its name. In addition, because each of the attributes is independent of the others, it is possible to retrieve or modify only selected attributes by performing a ***projection*** of the relation. Attributes may also be defined with ***keys***. If an attribute is keyed, an index structure is built that allows rapid direct access to a given entry. There is a restriction, however, that a keyed attribute must have unique values for all entries.

Another powerful feature is the ability to retrieve entries that have been qualified by one or more ***conditions***. A condition is a constraint definition for an attribute value. For instance, in the example above, the condition of X=1.0 might be specified prior to data retrieval. Only those entries that satisfy the given constraint or constraints are then returned.

Relational entities are used when the data they contain will be accessed or modified on a selective basis. This eliminates the need to move large sequential sets of data back-and-forth when modifying or retrieving only small amounts of data. An additional feature available with CADDB is the "blast" access of a relation. This allows the data to be treated sequentially while maintaining the relational form. these and other features are fully described in Section 8.5.

Matrix Entities.

One of the most important data structures encountered in engineering applications is the matrix. Matrix algebra forms the basis for the finite element method employed by ASTROS. The efficient performance of this algebra, along with additional operations such as simultaneous equation solvers, eigensolvers and integration schemes, is critical to such a software system. CADDB represents matrices in ***packed*** format. This format has been used extensively by the NASTRAN system for the last 30 years with excellent success. The representation of a matrix on CADDB is shown below:



1	ROW	n	...	ROW	n	...	END
2	ROW	n	...	END			
3	ROW	n	...	END			
4	ROW	n	...	ROW	n	...	END
5	ROW	n	...	END			

Referring to the figure, note that only the non-null columns of a matrix are stored, thus reducing disk space utilization. Within each column are one or more ***strings***. A string is a sequential burst of data

entities with a header that indicates the first row position of the data in the given column and "n", the number of terms in the string. This representation allows a further data compression in that zero terms in the column are not physically stored.

A complete library of matrix utilities is available within the ASTROS system. These utilities are coded to use the packed format to its best advantage. All matrix data should be stored in this manner. Many access methods are available for matrix entities. A matrix may be positioned randomly to a given column, an entire column may be read or written, individual terms may be read or written and so on. These functions are described in Subsection 8.4.

8.1. CADDB BASIC DESIGN CONCEPTS

The CADDB implementation was influenced by three fundamental design goals: Open-endedness, performance, and structured programming methodology. The basic internal design of the database imposes no unrealistic restrictions. A virtually unlimited number of different databases and entities can be processed simultaneously as long as there is memory to hold the required information. The only fixed restrictions are those imposed by the computer hardware and not the design. For example, the maximum number of blocks in the entire database is 2^{31} and the maximum number of words in each entity is also 2^{31}. This restriction follows from the 32-bit word length of some of the target machines.

The performance goals of the database had to address both I/O and CPU issues. The optimization of I/O performance is usually in direct conflict with minimal memory utilization. When faced with an I/O versus memory conflict, reduced I/O was generally selected. Summarized in the following are typical design decisions impacted by this issue:

1. All bit maps required by the database are kept in memory to reduce I/O requirements of free block management.

2. Directory pointers for all entities, open or closed, are kept in memory to reduce directory search time.

3. While any type of entity is open, all schema definition data are kept in memory.

CADDB was designed in top-down structured manner. It is divided into functional modules that simplify implementation, testing, and maintenance. Generically, the functions of these modules are:

1. ENTITY CODE: Separate groups of routines are provided for each of the three entity types.

2. RELATIVE BLOCK: These routines process the block allocation tables to convert relative block numbers used by entity routines into physical blocks.

3. BUFFER MANAGEMENT: All buffer management is done by these routines.

4. FREE BLOCK MANAGEMENT: Performs the allocating and freeing of physical blocks.

5. INDEX PROCESSING: All index processing is done by these routines. Two sets of routines, one for sequential indices and one for binary indices, are provided.

6. DIRECTORY: A separate set of routine is provided to do all directory processing.

7. MEMORY MANAGEMENT: All memory management is provided by these routines.

This highly modular design provides several advantages. The most important is that new features can be added with a minimal effect on the existing code. For example, a double buffering scheme could be added to reduce I/O wait time by simply modifying the buffer management routines.

8.1.1. Physical Structure

Each physical database is comprised of a set of disk files. An index file and at least one data file are required for each database. The index file contains the necessary control information to find entities on the database. This information includes the following:

1. DIRECTORY: Contains information required to process each entity.
2. FBBM: The Free Block Bit Maps (FBBM) are used to keep track of the blocks which are allocated and free.
3. BAT: The Block Allocation Table (BAT) is used to keep track of the physical blocks used by each entity.
4. SCHEMA: The SCHEMA defines the attribute structure for each relational entity.
5. INDEX: Each matrix or unstructured entity can have optional indices built to allow quick access to any column or record. Relational entities can also have indices built for any attribute.

The data files are used to store the actual information in each entity. Multiple data files can be used to split the database over several physical disk drives. Free block allocation is performed in a cyclic fashion among the data files to balance the I/O load on the system.

8.1.2. Improvements Over Other Databases

The design of a new ASTROS database was required to address deficiencies in existing available codes. The GINO I/O system of NASTRAN, while efficient, is a file management system, not a database. Separate files are required for each entity and only matrix and unstructured entities are supported. The RIM database, developed by the IPAD program for NASA, supports the relational entity type but does not adequately support either unstructured or matrix types. Additionally, the RIM system suffers from severe restrictions and performance penalties. The following summarize the functional improvements that make CADDB superior to these existing systems:

1. The three entity types have been combined into one database in as consistent a manner as possible.
2. The dynamic memory manager (See Subsection 8.3) allows the database to be open-ended without overburdening an application code which also makes large demands on memory.
3. Multiple databases and as many entities as memory allows may be processed simultaneously.
4. Multiple jobs can have READONLY access to the same database. With CADDB, a system database, as described in Chapter 3.2, is provided. This database contains data required by each ASTROS job.
5. An improvement over GINO allows existing records or columns of unstructured and matrix entities to be rewritten without destroying any other data in the entity.
6. "Garbage-collection" of freed blocks is handled automatically by the database. The dump and restore requirement of some databases, such as RIM, is eliminated.

7. The concept of projections has been added to all relational entity access calls. This allows application codes to process only those attributes needed for each entry of the relation. This allows a new attribute to be added to a relation without impacting previously coded modules not requiring the new attribute.

8.1.3. Memory Requirements

As discussed in the introduction to this section, trade-offs in design between memory and I/O performance were generally made in favor of I/O. In this subsection, the general memory requirements of CADDB are summarized. The equations below use the following symbols:

I	the index block file size in words
D	the data file block size in words
E	the number of entities on all open databases
P	the number of physical files in the database
N	the number of attributes for a relation

The following memory is required:

1. For the entity name table: $M_1 = 10E$
2. For each open database: $M_2 = 21 + 6P + I(P+1)$
3. For each open entity without indexing: $M_3 = 40 + D + I$
4. For each open entity with indexing: $M_4 = 40 + D + 31$
5. For each relation, an additional requirement is: $M_5 = 49N$

Using an index file block size of 256 words and a data file block size of 2,048 words, these relations indicate that, for a typical engineering module, the memory requirement would be approximately 4,000 words greater than that required by the NASTRAN GINO system.

This is felt to be a small trade-off for the significant capability enrichment.

8.2. THE GENERAL UTILITIES

There are nine general CADDB utility routines as shown below:

SUBROUTINE	FUNCTION
DBCREA	Creates a database entity
DBOPEN	Opens a database entity prior to I/O
DBRENA	Renames a database entity
DBEQUV	Equivalences two entity names
DBSWCH	Interchanges the names of two entities
DBDEST	Destroys, or removes, an entity and all of its data from the database
DBFLSH	Removes the data contents of an entity
DBCLOS	Terminates I/O for an entity
DBEXIS	Checks for existence of an entity
DBNEMP	Checks for existence of data in an entity

General Utilities are those which apply to any entity type. Two additional general data utilities are **DBINIT** and **DBTERM**. These are system level modules and are presented in Chapter 4.

Creating a New Entity.

To create a new database entity, the routine **DBCREA** is used. This utility enters the new entity name and its type into the database directory. Although there are three entity classes, there are two options for both matrix and unstructured entities, indexed or unindexed. Typical calls to create the three entities pictured in Subsection 8.1 could be:

```
CALL DBCREA ('GRID', 'REL')
CALL DBCREA ('STUF', 'IUN')
CALL DBCREA ('MATX', 'MAT')
```

The ASTROS executive system automatically creates all database entities that are declared in the MAPOL program. An application programmer usually creates only scratch entities within a given module.

Accessing Entities.

Prior to adding new data, modifying existing data or accessing old data for an entity, the entity must be opened, and when I/O is completed it must be closed. This is done to allow optional use of memory resources as discussed in Subsection 8.1. Using the examples as before, I/O is initiated by the calls:

```
CALL DBOPEN ('GRID', INFO, 'R/W', 'FLUSH', ISTAT)
CALL DBOPEN ('STUFF', INFO, 'RO', 'NOFLUSH', ISTAT)
CALL DBOPEN ('MAXT', INFO, 'R/W', 'FLUSH', ISTAT)
```

The array **INFO** is very important. It contains 20 words that provide information about the data contents of the entity, such as the number of attributes and entries in a relation, the number of records in an unstructured entity and the number of columns in a matrix. The first 10 words of **INFO** are used by the database. The programmer may use the second 10 words for any purpose desired. The **INFO** array is

then updated when the entity is closed. As an option, access to an entity may request that the data contents of the entity be destroyed, or **FLUSH**ed, when opening it.

When all activity is completed for a given entity, it must be closed to free memory used for I/O. This is done with a call such as:

```
CALL DBCLOS ('GRID')
```

Database General Utility Module: **DBCLOS**

Entry Point: **DBCLOS**

Purpose:

To terminate I/O from a specified database entity.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBCLOS ( ENTNAM )
```

ENTNAM	The name of the entity (Character, Input)

Method:

None

Design Requirements:

None

Error Conditions:

None

Database General Utility Module: **DBCREA**

Entry Point: **DBCREA**

Purpose:

To create a new data entity.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBCREA ( ENTNAM, TYPE )
```

ENTNAM	The name of the entity (Character, Input)
TYPE	The entity type (Character, Input)

'REL'	Relation
'MAT'	Matrix
'IMAT'	Indexed matrix
'UN'	Unstructured
'IUN'	Indexed unstructured

Method:

None

Design Requirements:

None

Error Conditions:

None

Database General Utility Module: **DBDEST**

Entry Point: **DBDEST**

Purpose:

To destroy a database entity, removing all data from the database files and the entity name from the list of entities.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBDEST ( ENTNAM )
```

ENTNAM The name of the entity (Character, Input)

Method:

None

Design Requirements:

1. **ENTNAM** may not be open.

Error Conditions:

None

Database General Utility Module: **DBEQUV**

Entry Point: **DBEQUV**

Purpose:

To equivalence two entity names to point to the same data. After a **DBEQUV** operation, the two names are synonymous. The only way to break an established equivalence is to destroy one of the entities which destroys the equivalences along with the entity and its data.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBEQUV ( NAME1, NAME2 )
```

NAME1	Name of currently existing entity (Character, Input)
NAME2	Name to be made equivalent to **NAME1** (Character, Input)

Method:

None

Design Requirements:

None

Error Conditions:

None

Database General Utility Module: **DBEXIS**

Entry Point: **DBEXIS**

Purpose:

To determine if a given entity already exists on the database.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBEXIS ( ENTNAM, EXIST, ITYPE )
```

ENTNAM	The name of the entity (Character, Input)	
EXIST	Status of the entity (Integer, Output)	
	0	does not exist
	1	exists
ITYPE	The entity type (Integer, Output)	
	0	undefined entity
	1	relation
	2	matrix
	3	indexed matrix
	4	unstructured
	5	indexed unstructured

Method:

None

Design Requirements:

None

Error Conditions:

None

Database General Utility Module: **DBFLSH**

Entry Point: **DBFLSH**

Purpose:

To delete, or flush all of the data from a database entity. The entity itself remains in existence, but is empty.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBFLSH ( ENTNAM )
```

ENTNAM	The name of the entity (Character, Input)

Method:

None

Design Requirements:

1. **ENTNAM** may not open.

Error Conditions:

None

Database General Utility Module: **DBNEMP**

Entry Point: **DBNEMP**

Purpose:

To return a logical TRUE or FALSE depending on whether an entity has entries, records or columns (TRUE) or if it is nonexistent or empty (FALSE).

MAPOL Calling Sequence:

None

Application Calling Sequence:

DBNEMP (ENTNAM)

ENTNAM	The name of the entity (Character, Input)

Method:

DBNEMP is a LOGICAL FUNCTION that returns **TRUE** if and only if the named **ENTNAM** exists and contains entries if relational, columns if matrix or records if unstructured. Any other condition returns a **FALSE**.

Design Requirements:

1. **ENTNAM** may not open.

Error Conditions:

None

Database General Utility Module: **DBOPEN**

Entry Point: **DBOPEN**

Purpose:

To open a database entity for subsequent I/O operations.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBOPEN ( ENTNAM, INFO, RW, FLUSH, ISTAT )
```

ENTNAM The name of the entity (Character, Input)

INFO Array of length 20 words containing entity information (Integer, Output)

INFO	RELATION	MATRIX	UNSTRUCTURED
1	TYPE	TYPE	TYPE
2	NATTR	NCOL	NREC
3	NENTRY	NROW	MAX REC LEN in words
4	–	PREC	—
5	–	DEN*100	—
6	–	FORM	—
7	–	Maximum number of nonzero terms in any column	—
8	–	Maximum number of strings in column	—
9	–	Maximum length of a string	—
10	–	—	—

DEN is measured in percent ranging from 0.0 to 100.0

Type Codes (TYPE) are:	
1	REL
2	MAT
3	IMAT
4	UN
5	IUN

Form Codes (FORM) are:	
1	rectangular
2	symmetric
3	diagonal
4	identity
5	square

Precision Codes (PREC) are:	
1	real, single-precision
2	real, double-precision
3	complex, single-precision
4	complex, double-precision

RW Type of access (Character, Input)

'R/W' Read/Write access
'RO' Read only access

FLUSH Flush option (Character, Input)

'FLUSH' flush entity on open
'NOFLUSH' do not flush entity on open

ISTAT Return status (Integer, Output)

0 entity opened
101 entity does not exist

Method:

None

Design Requirements:

1. The **INFO** array is loaded on the call to **DBOPEN** and not subsequently modified. The programmer may use the second 10 words for any purpose. **DBCLOS** will write the current **INFO** data to the database.
2. Multiple open entities must not share **INFO** array locations. Care must be taken not to modify the first 10 words within the application.

Error Conditions:

None

Database General Utility Module: **DBRENA**

Entry Point: **DBRENA**

Purpose:

To rename a database entity.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBRENA ( OLDNAM, NEWNAM )
```

OLDNAM	Existing entity name (Character, Input)
NEWNAM	New entity name (Character, Input)

Method:

None

Design Requirements:

1. The entity may be open.

Error Conditions:

None

Database General Utility Module: **DBSWCH**

Entry Point: **DBSWCH**

Purpose:

To switch the names of two database entities.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL DBSWCH ( NAME1 , NAME2 )
```

NAME1	Name of first entity (Character, Input)
NAME2	Name of second entity (Character, Input)

Method:

None

Design Requirements:

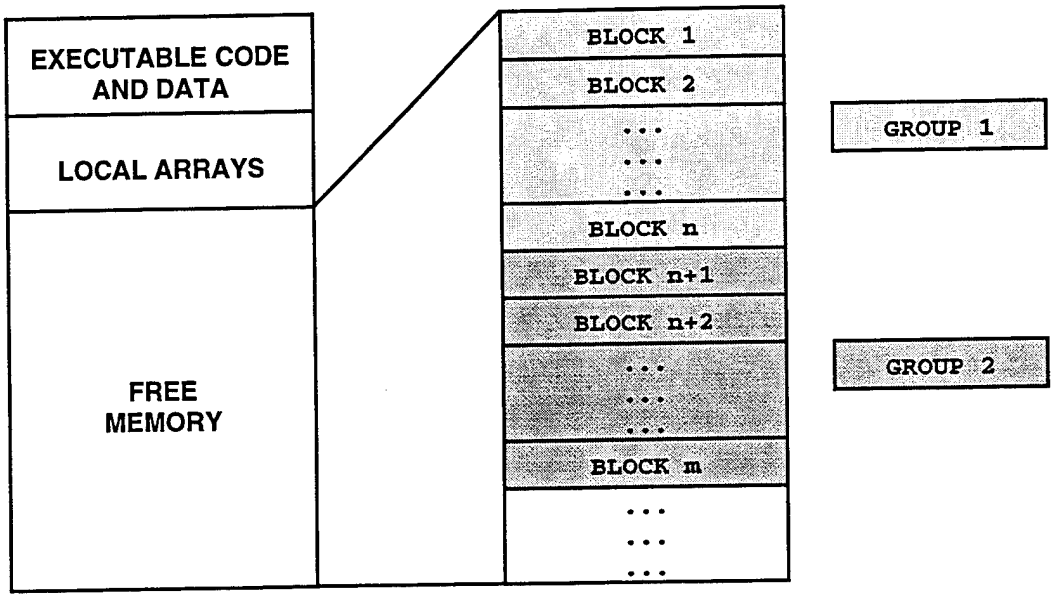
None

Error Conditions:

None

8.3. THE DYNAMIC MEMORY MANAGER UTILITIES

The dynamic memory manager (DMM) is a group of utility routines that allow the applications programmer to work with open-ended arrays in memory. This is important for two reasons. The first is that memory is not wasted by fixed length FORTRAN arrays. The second reason is to allow algorithms to use spill-logic. Spill-logic is code that can perform operations on only that portion of the required data that fits in memory at a given time. Free memory, (that beyond the fixed code and data areas) may be organized into any number of blocks, and groups of blocks, as shown below:



This dynamic memory area can be viewed as a type of virtual memory those paging is under control of the programmer. Static memory languages, such as FORTRAN, are very inefficient users of memory. The DMM can eliminate some of this inefficiency. As an example, consider a routine to perform the matrix addition

$$[A] + [B] = [C]$$

defined by the equations

$$C_{ij} = A_{ij} + B_{ij}$$

Three possible implementations are shown, all based on the assumption that the available memory, after all other components of the program are loaded, is 30,000 words.

The Classical FORTRAN Approach.

The classical brute-force FORTRAN solution to this problem is to see that 3 arrays each dimensions 100 by 100 will fit perfectly in the available memory. The routine is duly coded as:

```
      DIMENSION A (100, 100), B (100, 100), C (100, 100)
C
C  ASSUME THE MATRICES ARE ALL N*M
C
      DO 200 I=1, N
            DO 100 J=1, M
                C(I,J) = A(I,J) + B(I,J)
100         CONTINUE
200   CONTINUE
```

With this algorithm, the matrix sizes are fixed at 100 by 100. If the matrices are only 3 by 3, 99 plus percent of the memory is wasted. Further, although a 20 by 500 matrix would occupy the same 10,000 words, it cannot fit into the predefined array. This latter problem can easily be fixed by storing the matrix in a singly dimensioned array of 10,000, which already implies the programmer must manage the array.

By using the dynamic memory manager both problems shown in the last section disappear. Consider the code segment:

```
      COMMON/MEMORY/ Z (1)
C
C  ALLOCATE MEMORY FOR EACH MATRIX
C
      CALL MMBASE ( Z )
      CALL MMGETB ( 'AMAT', 'RSP', N*M, 'MAXT', IA, ISTAT )
      CALL MMGETB ( 'BMAT', 'RSP', N*M, 'MAXT', IB, ISTAT )
      CALL MMGETB ( 'CMAT', 'RSP', N*M, 'MAXT', IC, ISTAT )
      DO 100 I = 1, N*M
            II = I - 1
            Z ( IC + II ) = Z ( IA + II ) + Z ( IB + II )
100         CONTINUE
```

This code allows all 30,000 words of memory to be used regardless of the shape of the matrices. Additionally, it uses exactly the memory required if the operation is smaller than the available memory.

The Spill-logic Approach.

Spill-logic can be implemented in a number of ways using the matrix utilities described in Subsection 8.4. When spill-logic is used, only those portions of the matrices involved in an operation are brought into memory. Operations are then performed and intermediate or final results stored on the database. With this coding technique, problems of virtually unlimited size may be addressed. There are nine DMM utilities that may be used by an application programmer. Each routine is prefixed with the letters MM. A summary of these routines is shown below.

SUBROUTINE	FUNCTION
MMBASE MMBASC	Used by each module to define the location of the memory base address
MMDUMP	Prints a table of allocated memory blocks
MMFREE MMFREG	Frees allocated memory by individual blocks or by groups of blocks
MMGETB	Gets a block of memory of the specified type and length
MMREDU	Reduces the size of a block
MMSQUZ	Compresses memory I/O areas
MMSTAT	Returns the maximum contiguous memory that is available to the module

Database Memory Manager Utility Module: **MMBASC**

Entry Point: **MMBASC**

Purpose:

To define the base address of dynamic memory that contains character data.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MMBASC ( ARAY, LEN )
```

ARAY	The name of a character array from which memory pointers will be measured
LEN	The length of the character string elements in **ARAY**

Method:

None

Design Requirements:

1. Only one call to **MMBASC** may be made in a module.

Error Conditions:

None

NoneDatabase Memory Manager Utility Module: **MMBASE**

Entry Point: **MMBASE**

Purpose:

To define the base address of dynamic memory of reference to an array.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MMBASE  ( ARAY )
```

ARAY The name of an array from which memory pointers will be measured

Method:

None

Design Requirements:

1. This routine must be the first called in each module that uses the memory manager.
2. It cannot be used for memory containing character data (see **MMBASC**).

Error Conditions:

None

Database Memory Manager Utility Module: **MMDUMP**

Entry Point: **MMDUMP**

Purpose:

To print a formatted table of allocated memory blocks to the output file.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MMDUMP
```

Method:

None

Design Requirements:

1. The utility assumes the **MMINIT** has been called to initialize the open core memory block.
2. In some cases of corrupted memory, the use of this routine may result in an infinite loop.

Error Conditions:

None

Database Memory Manager Utility Module: **MMFREE**

Entry Point: **MMFREE**

Purpose:

To free a memory block for subsequent use.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MMFREE ( BLK )
```

BLK Name of block to be freed (Character, Input)

Method:

None

Design Requirements:

None

Error Conditions:

None

Database Memory Manager Utility Module: **MMFREG**

Entry Point: **MMFREG**

Purpose:

To

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MMFREG ( GRP )
```

GRP	Name of the group of blocks to be freed (Character, Input)

Method:

None

Design Requirements:

None

Error Conditions:

None

Database Memory Manager Utility Module: **MMGETB**

Entry Point: **MMGETB**

Purpose:

To allocate a block of dynamic memory.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MMGETB ( BLK, TYPE, LEN, GRP, IPNT, ISTAT )
```

BLK — The name assigned to this memory block
If the block name is blank, the block is a special unnamed memory block in that it can only be freed with a **MMFREG** call and **MMREDU** calls are not allowed for the block. (Character, Input)

TYPE — The data type of the memory block: (Character, Input)

'RSP'	real, single-precision or integer
'RDP'	real, double-precision
'CSP'	complex, single-precision
'CDP'	complex, double-precision
'CHAR'	character

LEN — Length of block measured in the units of **TYPE** (Integer, Input)

GRP — Name defining a group to which this block belongs (Character, Input)

IPNT — Pointer to the allocated block of memory referenced to the base location (Integer, Output)

ISTAT — Status return

0	memory successfully allocated
101	insufficient memory available

Method:

None

Design Requirements:

1. The **BLK** and **GRP** names are truncated to a length of four characters. Thus, the names should be unique for these characters.

Error Conditions:

1. Attempt to allocate a memory block of zero length.
2. Attempt to allocate a duplicate block/group name.

Database Memory Manager Utility Module: **MMREDU**

Entry Point: **MMREDU**

Purpose:

To reduce the size of a memory block that is larger than needed.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MMREDU    ( BLK, TYPE, LENGTH )
```

BLK	The name assigned to this memory block (Character, Input)
TYPE	The data type of the memory block: (Character, Input)
	'RSP' real, single-precision or integer
	'RDP' real, double-precision
	'CSP' complex, single-precision
	'CDP' complex, double-precision
	'CHAR' character
LENGTH	Length to be freed measured in units of **TYPE** (Integer, Input)

Method:

None

Design Requirements:

None

Error Conditions:

1. Attempt to reduce memory block to zero length.

Database Memory Manager Utility Module: **MMSQUZ**

Entry Point: **MMSQUZ**

Purpose:

To squeeze, or compress, any unused I/O buffer space used by the database.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MMSQUZ
```

Method:

None

Design Requirements:

1. This routine should not be used within an application routine, it is an executive memory management function.

Error Conditions:

None

Database Memory Manager Utility Module: **MMSTAT**

Entry Point: **MMSTAT**

Purpose:

To determine the maximum number of contiguous single-precision words available for dynamic allocation.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MMSTST ( CONTIG )
```

CONTIG	The maximum number of contiguous single-precision words available (Integer, Output)

Method:

None

Design Requirements:

None

Error Conditions:

None

8.4. UTILITIES FOR MATRIX ENTITIES

The matrix entity utilities are designed to provide a number of different methods for accessing complete columns, portions of columns, or single terms of a matrix. The use of these various methods depends on the source of data defining the matrix and the intensity of the computational algorithm. The routines available are:

SUBROUTINE	FUNCTION
MXINIT	Initializes a matrix entity for I/O
MXFORM	Change the form of a matrix
MXPOS	Positions to a specified matrix column
MXRPOS	
MXNPOS	
MXSTAT	Gets matrix column information
MXPAK	Packs a column of a matrix
MXUNP	Unpacks a column of a matrix
MXPKTI	Packs a column of a matrix either term-by-term or by partial column
MXPKT	
MXPKTM	
MXPKTF	
MXUPTI	Unpacks a column of a matrix either term-by-term or by partial column
MXUPT	
MXUPTM	
MXUPTF	

8.4.1. Creating a Matrix.

After a matrix entity has been created it must be initialized before it can be used. The **MXINIT** call provides information required for the storing of data into the matrix. For example, to create and initialize a matrix entity for a real, single-precision, symmetric matrix with 1,000 rows the following code is required:

```
INTEGER INFO (20)
CALL DBCREA ('KGG', 'MAT')
CALL DBOPEN ('KGG', INFO,'R/W', 'FLUSH', ISTAT)
CALL MXINIT ('KGG', 1000, 'RSP', 'SYM')
```

Whenever a matrix is flushed, with either a **DBOPEN** or a **DBFLSH** call, the initialization data are cleared. Therefore, an **MXINIT** call is required before reusing the matrix in this case. Similarly, if a matrix entity is going to be redefined, it must be flushed before a new **MXINIT** call may be made.

8.4.2. Packing and Unpacking a Matrix by Columns.

The simplest method to process a matrix is with the full column routines **MXPAK** and **MXUNP**. Each of these routines may process either a full column or a portion of a column. In either case, only one call is allowed for each column. The subsequent call will process the next column. The following code illustrates the packing and unpacking of matrix by columns:

```
C
C         PACK MATRIX BY COLUMNS
C
          DO 100 ICOL = 1, NCOL
              CALL MXPAK ('KGG', COLDTA(1,ICOL),1,1000)
100       CONTINUE
C
C         UNPACK MATRIX BY COLUMNS
C
          CALL MXPOS ('KGG',1)
          DO 200 ICOL = 1, NCOL
              CALL MXUNP ('KGG', DATA(1,ICOL),1,1000)
200       CONTINUE
```

8.4.3. Obtaining Matrix Column Statistics.

The **MXPAK** routine removes any zero terms in the column to reduce the amount of disk space required to store the matrix. Consecutive nonzero terms are stored in strings. Whenever a zero term is encountered, the current string is terminated and a new string is started. The **MXSTAT** routine may be used to obtain statistics about each column. The following code gives an example of how this information can be used to unpack only those terms between the first and last nonzero terms.

```
          DO 100 ICL = 1, NCOL
              CALL MXSTAT ('KGG', COLID, FNZ, LNZ, NZT, DEN, NSTR)
              CALL MXUNP ('KGG', DATA, FNZ, LNZ-FNZ + 1)
100       CONTINUE
```

8.4.4. Packing and Unpacking a Matrix by Terms.

A matrix can also be processed by individual terms. To pack a matrix termwise requires a series of calls for each column. The first call must be a column initialization call, followed by a series of calls to pack single terms and, finally, a column termination call. The following code shows the packing of an individual matrix column by terms:

```
C
C         INITIALIZE TERM-WISE PACKING
C
          CALL MXPKTI ('KGG', IKGG)
C
C         READ MATRIX TERM AND PACK
C
100       READ (5, *, END=200) IROW, VAL
          CALL MXPKT (IKGG, VAL, IROW)
          GO TO 100
C
C         TERMINATE COLUMN
C
200       CALL MXPKTF ('KGG')
```

Note that the termwise packing must be done in ascending row order.

A similar set of calls is required to unpack a matrix by terms. The **MXSTAT** routine is used to determine the number of nonzero terms that exist in the column. The following code will unpack and print the nonzero terms for a matrix column.

```
C         DETERMINE NUMBER OF TERMS
C
C
          CALL MXSTAT ('KGG', COLID, FNZ, LNZ, NZT, DEN, NSTR)
C
C         START UNPACKING THE MATRIX COLUMN
C
          CALL MXUPTI ('KGG', IKGG)
          DO 100 ITERM=1, NZT
              CALL MXUPT (IKGG, VAL, IROW)
              WRITE (6,*) 'ROW=', IROW, 'TERM=',VAL
100       CONTINUE
C
C
C
          CALL MXUPTF ('KGG')
```

It is not required that each term in the column be unpacked. If any terms are left, the **MXUPTF** routine will ignore them and position the matrix to the next column.

8.4.5. Packing and Unpacking a Matrix by Strings.

As explained earlier, matrix data are actually stored in strings of terms with intervening zero terms compressed. A series of routines is provided to allow matrices to be accessed by strings. The use of these routines is similar to the termwise routines in that there is a column initialization call, a call for each string, and a column termination call. The following code shows the packing of a matrix which contains two strings, the first with five terms and the second with three terms.

```
C
C         INITIALIZE FOR STRING PACKING
C
          CALL MXPKTI ('KGG', IKGG)
C
C         PACK OUT TWO STRINGS
C
          CALL MXPKTM (IKGG, STR1, 10, 5)
          CALL MXPKTM (IKGG, STR2, 20, 3)
C
C         TERMINATE STRING PACKING
C
          CALL MXPKTF ('KGG')
```

Packing a column by strings differs in several respects from packing by columns. First, more than one **MXPKTM** call is allowed for each column. With **MXPAK** only one call per column is allowed. Secondly, no compression of zero terms is done within a string. This feature can be used to insure that certain terms of a matrix are stored, even if zero, so they can later be rewritten randomly.

The unpacking of a matrix by strings is shown in the following code example. Note the use of the **MXSTAT** routine to determine the number of strings stored in the column.

```
C
C         DETERMINE THE NUMBER OF STRINGS
C
          CALL MXSTAT ('KGG', COLID, FNZ, LNZ, NZT, DEN, NSTR)
C
C         UNPACK COLUMN BY STRINGS
C
          CALL MXPKTI ('KGG', IKGG)

          DO 100 I=1, NSTR
              CALL MXUPTM ('KGG', VALS, IROW, NROW)
              WRITE (6,*) 'TERMS FROM ROW', IROW, 'TO ROW',
         * IROW+NROW-1, 'ARE', (VALS (I), I=1,NROW)
100       CONTINUE
C
C         TERMINATE COLUMN UNPACKING
C
          CALL MXUPTF ('KGG')
```

It is important that **MXSTAT** be used to determine the number of strings in a column because, under several conditions, the number may be different from the number of strings packed. First, if the

string packed does not fit in the current buffer, it will be automatically split over two buffers. Secondly, if two strings are packed consecutively, they will be automatically merged into one string in the buffer.

8.4.6. Matrix Positioning.

Several routines are provided to position a matrix randomly to a given column. This operation can be done on either indexed, **IMAT**, or unindexed matrices, but the presence of an index speeds up the processing greatly. The following code shows the use of these routines to randomly read three matrix columns.

```
C
C       POSITION TO COLUMN 10 AND UNPACK
C
        CALL MXPOS ('KGG', 10)
        CALL MXUNP ('KGG', DATA, 1, 1000)
C
C       POSITION FORWARD 5 COLUMNS
C
        CALL MXRPOS ('KGG', +5)
        CALL MXUNP ('KGG', DATA , 1, 1000)
C
C       POSITION TO NEXT NONNULL COLUMN
C
        CALL MXNPOS ('KGG', ICOL)
        CALL MXUNP ('KGG', DATA, 1, 1000)
```

The first **MXUNP** retrieves the data for column 10 and leaves the matrix positioned at the start of column 11. The **MXRPOS** call positions the matrix forward five columns to column 16. The second **MXUNP** call then retrieves the data for column 16. The results of the **MXNPOS** call depend on the data stored in the matrix. If column 17 has nonzero terms, it will be positioned there. If column 17 is null, the matrix will be positioned forward until a nonnull column is found. Note that both **MXPOS** and **MXRPOS** require that the column to which the matrix is positioned exists. The **MXNPOS**, utility is the more general in that it determines the next column that exists. Null matrix columns can be packed in two ways. The following code gives examples which are quite different in that the first example creates a column with no nonzero terms while the second example creates a null column which does not exist.

```
C
C       CREATE A COLUMN OF ZEROS WITH MXPAK
C
        CALL MXPAK ('KGG', 0.0,1,1)
C
C       CREATE NULL COLUMN
C
        CALL MXPKTI ('KGG', IKGG)
        CALL MXPKTF ('KGG')
```

8.4.7. Missing Matrix Columns.

For extremely sparse matrices it is possible to pack only the columns which contain data. This feature can greatly reduce disk space requirements for these matrices because space is not wasted for column headers and trailers. It also simplifies coding because it is not required to pack null columns. The following example shows the packing of an extremely sparse matrix which contains only two items.

```
C
C         PACK DIAGONAL TERM IN COLUMN 100
C
          CALL MXPOS ('KGG', 100)
          CALL MXPAK ('KGG', 1.0,100,1)
C
C         PACK DIAGONAL TERM IN COLUMN 500
C
          CALL MXPOS ('KGG', 500)
          CALL MXPAK ('KGG', 1.0,100,1)
```

When a matrix does not have all its columns stored, care must be used when unpacking it. Since the routines only operate on columns physically stored in the matrix only two sets of calls are required to unpack the matrix.

The following code example shows one method of unpacking this matrix.

```
C
C         UNPACK TWO MATRIX COLUMNS
C
          DO 100 ICOL = 1,2
              CALL MXSTAT ('KGG', COLID, FNZ,LNZ,NZT,DEN,NSTR)
              WRITE (6,*) 'DATA FOR COLUMN', COLID
              CALL MXUNP ('KGG', DATA,1,1000)
100       CONTINUE
```

This example illustrates a disadvantage. The code must know the exact number of columns stored in the matrix. There is no method provided to determine this. The next example shows how **MXNPOS** can be used to produce a code sequence that will work no matter how many physical columns are stored in the matrix.

```
C
C         POSITION TO NEXT COLUMN
C
100       CALL MXNPOS ('KGG',ICOL)
          IF ( ICOL.GT.0 ) THEN
              WRITE (6,*) 'DATA FOR COL', ICOL
              CALL MXUNP ('KGG', DATA,1,1000)
              GO TO 100
          ENDIF
```

The **MXPOS** and **MXRPOS** utilities should be used with extreme caution if the matrix does not contain all physical columns. These routines work on actual column numbers and will cause fatal errors if the column does not exist. For example, an **MXPOS** to column 200 will cause an error because the column

is not stored in the matrix. If the matrix is positioned at column 100, an **MXRPOS** of +100 will also fail because column 200 is not stored in the matrix.

8.4.8. Repacking a Matrix.

Once a matrix has been packed, it is possible to rewrite certain columns of the matrix without disturbing the data stored in any other columns. The only restriction is that the topology of the matrix terms cannot change. For example, if **MXPAK** was used to pack the column, all zero terms are compressed. Since these terms are not physically stored in the matrix, they cannot at a later time be replaced by a nonzero term. This can be avoided by using the termwise or stringwise calls, which perform no zero compression. The following example shows the packing of a matrix column and the subsequent repacking of it.

```
C
C         PACK COLUMN 1 OF MATRIX
C
          CALL MXPOS ('KGG',1)
          CALL MXPKTI ('KGG', IKGG)
          CALL MXPKTM (IKGG, STR, 10,10)
          CALL MXPKTF ('KGG')
C
C         READ COLUMN 1 OF MATRIX
C
          CALL MXPOS ('KGG',1)
          CALL MXPKTI ('KGG', IKGG)
          CALL MXPKTM (IKGG, DATA,IROW,NROW)
          CALL MXPKTF ('KGG')
C
C         DOUBLE EACH NUMBER IN THE STRING
C
          DO 100 I=1, NROW
              DATA (I)=DATA (I) * 2.0
100       CONTINUE
C
C         REPLACE THE STRING
C
          CALL MXPOS ('KGG',1)
          CALL MXPKTI ('KGG', IKGG)
          CALL MXPKTM (IKGG, DATA,IROW,NROW)
          CALL MXPKTF ('KGG')
```

All the matrix pack utility calls, i.e., column, term and string, may be used to repack matrix columns.

Database Matrix Utility Module: **MXFORM**

Entry Point: **MXFORM**

Purpose:

To change the form of a matrix entity.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MXFORM ( NAME, FORM )
```

NAME	The matrix name (Character, Input)
FORM	The new matrix form (Character, Input)

'REC'	Rectangular
'SYM'	Symmetric
'DIAG'	Diagonal
'INDENT'	Identity
'SQUARE'	Square

Method:

None

Design Requirements:

1. **MXFORM** may be called any time after the creation of the matrix.

Error Conditions:

1. Illegal **FORM** value; error **MXFORM01**.

Database Matrix Utility Module: **MXINIT**

Entry Point: **MXINIT**

Purpose:

To initialize a matrix prior to writing data.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MXINIT ( MATNAM, NROW, PREC, FORM )
```

MATNAM	Name of matrix (Character, Input)	
NROW	Number of rows (Integer, Input)	
PREC	The precision of the matrix (Character, Input)	
	'RSP'	Real, Single-precision
	'RDP'	Real, Double-precision
	'CSP'	Complex, Single-precision
	'CDP'	Complex, Double-precision
FORM	Form of the matrix (Character, Input)	
	'REC'	Rectangular
	'SYM'	Symmetric
	'DIAG'	Diagonal
	'INDENT'	Identity
	'SQUARE'	Square

Method:

None

Design Requirements:

None

Error Conditions:

None

Database Matrix Utility Module: **MXNPOS**

Entry Point: **MXNPOS**

Purpose:

To position a matrix to the next nonnull column.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MXNPOS ( MATNAM, ICOL )
```

MATNAM	Name of matrix (Character, Input)
ICOL	Column number positioned to (Integer, Output)

Method:

None

Design Requirements:

1. If there are no more nonnull columns in the matrix, **ICOL** is set to zero

Error Conditions:

None

Database Matrix Utility Module: **MXPAK**

Entry Point: **MXPAK**

Purpose:

To pack all , or a portion, of a matrix and then to move to the next column.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MXPAK ( MATNAM, ARAY, ROW1, NROW )
```

MATNAM	Name of matrix to be packed (Character, Input)
ARAY	Array containing data to be packed (Any type, Input)
ROW1	First row position in column (Integer, Input)
NROW	Number of rows to pack (Integer, Input)

Method:

None

Design Requirements:

None

Error Conditions:

None

Database Matrix Utility Module: **MXPKT**

Entry Point: **MXPKT**

Purpose:

To pack a column of a matrix one term at a time.

MAPOL Calling Sequence:

None

Application Calling Sequence:

CALL MXPKT (UNITID, VAL, IROW)

UNITID	Unit identification from **MXPKTI** call (Integer, Input)
VAL	The value to be packed (Any type, Input)
IROW	The row position of the term. **IROW** must be positive and greater than the value in any previous **MXPKT** call for the current column (Integer, Input)

Method:

None

Design Requirements:

None

Error Conditions:

None

Database Matrix Utility Module: **MXPKTF**

Entry Point: **MXPKTF**

Purpose:

To terminate the termwise or partial packing of a matrix column.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MXPKTF ( MATNAM )
```

MATNAM	Name of the matrix being packed (Character, Input)

Method:

None

Design Requirements:

None

Error Conditions:

None

Database Matrix Utility Module: **MXPKTI**

Entry Point: **MXPKTI**

Purpose:

To initialize a matrix column for term-by-term or partial packing.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MXPKTI ( MATNAM, UNITID )
```

MATNAM	Name of the matrix to be packed (Character, Input)
UNITID	Unit identifier (Integer, Output)

Method:

None

Design Requirements:

1. A matrix may be packed by columns using **MXPAK**, by terms, or by partial columns, but not by any combination.

Error Conditions:

None

Database Matrix Utility Module: **MXPKTM**

Entry Point: **MXPKTM**

Purpose:

To pack a column of a matrix using partial columns.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MXPKTM ( UNITID, VALARR, ROW1, NROW )
```

UNITID	Unit identifier from **MXPKTI** call (Integer, Input)
VALARR	Array of values to be packed (Any type, Input)
ROW1	Initial row position of **VALARR** (Integer, Input)
NROW	Number of rows to be packed (Integer, Input)

Method:

None

Design Requirements:

1. **ROW1** and **NROW** must be positive.

Error Conditions:

None

Database Matrix Utility Module: **MXPOS**

Entry Point: **MXPOS**

Purpose:

To position a matrix to a specified column.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MXPOS ( MATNAM, COL )
```

MATNAM	Name of the matrix (Character, Input)
COL	Column number (Input, Integer)

Method:

None

Design Requirements:

None

Error Conditions:

None

Database Matrix Utility Module: **MXRPOS**

Entry Point: **MXRPOS**

Purpose:

To position a matrix to a column by specifying the column increment relative to the current column.

MAPOL Calling Sequence:

None

Application Calling Sequence:

CALL MXPOS (MATNAM, DELCOL)

MATNAM	Name of the matrix (Character, Input)
DELCOL	Column number increment (Integer, Input)

Method:

None

Design Requirements:

1. Positive **DELCOL** positions forward, negative position backward from current column.

Error Conditions:

None

Database Matrix Utility Module: **MXSTAT**

Entry Point: **MXSTAT**

Purpose:

To obtain status information for the current column.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MXSTAT ( MATNAM, COLID, FNZ, LNZ, NZT, DEN, NSTR )
```

MATNAM	Name of the matrix (Character, Input)
COLID	Current column number (Integer, Output)
FNZ	First nonzero row in column(Integer, Output)
LNZ	Last nonzero row in column (Integer, Output)
NZT	Number of nonzero rows in column (Integer, Output)
DEN	Column density (Real, Output) (**DEN** is a decimal fraction, e.g., 40%=.40)
NSTR	Number of strings in column (Integer, Output)

Method:

None

Design Requirements:

1. Note that for very large matrices, **DEN** is a single-precision number and may be numerically zero even if there are nonzero terms in the matrix.

Error Conditions:

None

Database Matrix Utility Module: **MXUNP**

Entry Point: **MXUNP**

Purpose:

To unpack all, or a portion, of a matrix column and then to move to the next column.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MXUNP ( MATNAM, ARRAY, ROW1, NROW )
```

MATNAM	Name of the matrix (Character, Input)
ARRAY	Array containing data to be unpacked (Any type, Output)
ROW1	First row position in column (Integer, Input)
NROW	Number of rows to unpack (Integer, Input)

Method:

None

Design Requirements:

None

Error Conditions:

None

Database Matrix Utility Module: **MXUPT**

Entry Point: **MXUPT**

Purpose:

To unpack a column of a matrix one term at a time.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MXUPT ( UNITID, VAL, IROW )
```

UNITID	Unit identification from **MXUPTI** call (Integer, Input)
VAL	The value of the term (Any type, Output)
IROW	The row position of the term (Integer, Output)

Method:

None

Design Requirements:

None

Error Conditions:

None

Database Matrix Utility Module: **MXUPTF**

Entry Point: **MXUPTF**

Purpose:

To terminate the termwise or partial unpacking of a matrix column.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MXUPTF ( MATNAM )
```

MATNAM	Name of the matrix being unpacked (Character, Input)

Method:

None

Design Requirements:

None

Error Conditions:

None

Database Matrix Utility Module: **MXUPTI**

Entry Point: **MXUPTI**

Purpose:

To initialize a matrix column for term-by-term or partial unpacking.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MXUPTI ( MATNAM, UNITID )
```

MATNAM	Name of the matrix (Character, Input)
UNITID	Unit identifier (Integer, Output)

Method:

None

Design Requirements:

1. A matrix may be unpacked by columns using **MXUNP**, by term, or by partial columns, using **MXUPT** and **MXUPTM**, but not by any combination.

Error Conditions:

None

Database Matrix Utility Module: **MXUPTM**

Entry Point: **MXUPTM**

Purpose:

To unpack partial columns of a matrix.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL MXUPTM ( UNITID, VALARR, ROW1, NROW )
```

UNITID	Unit identifier from **MXUPTI** call (Integer, Input)
VALARR	Array that will contain the unpacked rows (Any type, Output)
ROW1	Initial row position being unpacked (Integer, Output)
NROW	Number of rows being unpacked (Integer, Output)

Method:

None

Design Requirements:

None

Error Conditions:

None

8.5. UTILITIES FOR RELATIONAL ENTITIES

Relational database entities are used to save highly structured data that will be accessed and modified in a random manner. Utilities to operate on relations are summarized below:

SUBROUTINE	FUNCTION
RESCHM	Defines the schema of a relation
REPROJ	Defines the projection of the relation prior to I/O activity
REQURY	Queries the schema of a relation
REGET	Gets, or fetches, a qualified entry from a relation
REGETM	
REUPD	Updates the current entry of a relation
REUPDM	
READD	Adds a new entry to a relation
READDM	
REPOS	Positions a relation to an entry
RECPOS	Checks for existence of a given entry
RECOND	Defines constraints or **WHERE** conditions for the relation
RESETC	
REENDC	
RENULD	Checks if an attribute has a **NULL** value
RENULI	
RENULR	
RENULS	
RECLRC	Clears conditions defined for a relation
REGB	Gets, or fetches, all of the qualified entries from a relation
REGBM	
REAB	Adds a group of entries to a relation
REABM	
RESORT	Sorts the entries of a relation

8.5.1. Examples of Relational Entity Utilities.

This subsection provides specific examples of operating with relations. Particular attention should be given the use of double-precision data attributes. Special routines are provided for such attributes when used by themselves or when "mixed" with other data types.

8.5.2. Creating a Relation.

A relational entity has both a name and a schema. The schema defines the attributes of a relation and their data types. Therefore, a call to the **RESCHM** routine is required in addition to a **DBCREA** call in order to complete the creation of a relational entity. For example, to create relation **GRID** (shown in the introduction to Section 8), the following code is required:

```
C
C        DEFINE ATTRIBUTES TYPES, AND LENGTHS
C
   CHARACTER *8       GATTR (4)
   CHARACTER *8       GTYPE (4)
   INTEGER            GLEN (4)
   DATA GATTR / 'GID', 'X', 'Y', 'Z'/
   DATA GTYPE / 'KINT', 'RSP', 'RSP','RSP' /
   DATA GLEN / 0,0,0,0 /
C
C  CREATE A RELATION AND SCHEMA
C
   CALL DBCREA ( 'GRID' , 'REL')
   CALL RESCHM ('GRID', 4, GATTR, GTYPE, GLEN )
```

The schema is specified by attribute name and data type. Various data types are available. In the example, the grid **ID**, **GID**, is called a keyed integer (**KINT**). This causes an index structure to be created that will allow fast direct access to a given entry. The coordinate values **X**, **Y**, and **Z** are defined as real, single-precision (**RSP**). The length parameters (**GLEN**) are only used for character attributes and for arrays of integers or real numbers. An array of values would be used if the overall data organization is relational but some groups of values are only used on an all-or-nothing basis.

8.5.3. Loading Relational Data.

Once a relation has been created it may be loaded with data. There are two modes of adding data: one entry at a time, or a "blast" add wherein the entire relation, or a large part of it, has been accumulated in memory. For each mode, there are two options, one when none of the attributes are real, double-precision, and a second if one or more attributes are real, double-precision. Using the relation **GRID**, the example below indicates how it could be loaded on an entry-by-entry basis.

```
C
C         ALLOCATE BUFFER AREA FOR ENTRIES AND INFO
C
          INTEGER IBUF (4), INFO (20)
C
C         USE EQUIVALENCES TO HANDLE REAL DATA
C
          EQUIVALENCE (IGID, IBUF (1)), (X, IBUF (2) )
          EQUIVALENCE (y, IBUF (3)), (Z, IBUF (4) )
C
C         DEFINE THE PROJECTION AS THE FULL RELATION
C
          CHARACTER *8 PATTR (4)
          DATA PATTR / 'GID', 'X', 'Y', 'Z'/
C
C         OPEN THE ENTITY FOR I/O
C
          CALL DBOPEN ('GRID', INFO, 'R/W', 'FLUSH', ISTAT)
          CALL REPROJ ('GRID', 4, PATTR)
C
C         READ AN ENTRY FROM INPUT, ADD TO RELATION
C
          DO 100 I=1, IEND
              READ (5, 101) IGID, X,Y,Z
              CALL READD ('GRID', IBUF)
100       CONTINUE
C
C
C
          CALL DBCLOS ('GRID')
```

Space must first be allocated to contain an entire entry of the relation. This buffer must be of a specific type so that equivalences must be used if the attributes are of mixed data types. The projection of the relation must be defined (via **REPROJ**) even if all attributes are being selected. If the data had been stored in memory first, the **READ** routine could have been used to "blast" all of the entries into the relation with a single call.

8.5.4. Accessing a Relation

A relation is accessed by a set of four routines: **REGET**, **REGETM**, **REGB**, and **REGBM**. Several other routines now come into play. The first are **REPOS** and **RECPOS**. These routines are used to find an entry within a relation whose key is equal to a specific value. The second is the group of routines **RECOND**, **RESETC**, **REENDC**, and **RECLRC**. These allow the specification of more complex "where" clauses that are used to qualify an entry of the relation.

As an example, suppose that the X, Y, and Z coordinates are to be retrieved for a grid point whose **GID** is 1. The code segment below could be used to perform this:

```
C
C         ALLOCATE BUFFER - ALL OUTPUT IS REAL
C
          DIMENSION COORDS (3), INFO (20)
C
C         DEFINE THE PROJECTION
C
          CHARACTER *8 PATTR (3)
          DATA PATTR / 'X', 'Y', 'Z'/
C
C         OPEN THE ENTITY FOR I/O
C
          CALL DBOPEN ('GRID', INFO, 'R/W', 'NOFLUSH', ISTAT)
          CALL REPROJ ('GRID', 3, PATTR)
C
C         POSITION TO THE DESIRED ENTRY
C
          CALL REPOS ('GRID', 'GID', 1)
C
C         GET THE ENTRY
C
          CALL REGET ('GRID', COORDS, ISTAT)
```

Note that **GID** must be a keyed attribute to use **REPOS**.

To qualify an entry by more than one attribute, a sequence of an **RECOND** call, any number of **RESETC** calls, and an **REENDC** call can be used. For instance, to find any or all grid points whose coordinates are X=1, Y=2, Z=3, the code segment below could be used:

```
CALL RECOND ('GRID', 'X', 'EQ', 1.0)
CALL RESETC ('AND', 'Y', 'EQ', 2.0)
CALL RESETC ('AND', 'Z', 'EQ', 3.0)
CALL REENDC
CALL REGET ('GRID', BUF, ISTAT)
```

Each call to **REGET** will retrieve an entry that satisfies the specified conditions. An **ISTAT** value greater than zero indicates the end of successful retrievals. Conditions may include any of the relational operators, the **MAX** and **MIN** selectors and the Boolean operators **AND** and **OR**. If one of either **MIN** or **MAX** issued, however, it is the only condition allowed. To reset a new set of conditions on an open relational entity, the utility **RECLRC** may be called to destroy the current conditions.

8.5.5. Updating a Relational entry.

One of the most powerful features of the relational database is the ability to randomly modify a small number of data items efficiently. To do this, the utilities **REDUPD** and **REDUPM** are used. The update procedure is a simple one. First, the projection is set. This is followed by positioning to a row or rows by specifying a **REPOS**, **RECPOS** or an **RECOND**. Routine **REGET** is then used to fetch the entry. One or more of

attributes may then be modified in the buffer and an **REDUP** used to accomplish the update. Note that attributes not in the projection, and attributes not modified in the buffer, will remain unchanged.

8.5.6. Other Operations.

If it is necessary for an application to determine the schema of a given relation, this may be done with the utility **REQURY**. This routine returns the names and types of each attribute in the schema. Finally, a relation may be sorted in an ascending or descending manner on one or more of its attributes by using the utility **RESORT**.

Database Relational Utility Module: **REAB**

Entry Point: **REAB**

Purpose:

To add multiple entries, held in memory, to a specified relation.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL REAB ( RELNAM, BUF, INUM )
```

RELNAM	Name of the relation (Character, Input)
BUF	Array that contains the entries to be added to the relation (Any, Input)
INUM	The number of entries to be added (Integer, Input)

Method:

None

Design Requirements:

1. Only integer, real single-precision, or string attributes may be added with this routine.

Error Conditions:

None

Database Relational Utility Module: **REABM**

Entry Point: **REABM**

Purpose:

To add multiple entries, held in memory, to a specified relation where the relational attributes are double-precision, or mixed precision, types.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL REABM ( RELNAM, SNGL, DBLE, INUM )
```

RELNAM	Name of the relation (Character, Input)
SNGL	Array that contains the single-precision (Integer, Real, Single-Precision, or String) attributes to be added (Any, Input)
DBLE	Array that contains the double-precision attributes to be added (Double, Input)
INUM	The number of entries to be added (Integer, Input)

Method:

None

Design Requirements:

None

Error Conditions:

None

Database Relational Utility Module: **READD**

Entry Point: **READD**

Purpose:

To add a new entry to a relation.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL READD ( RELNAM, BUF )
```

RELNAM	Name of the relation (Character, Input)
BUF	Array that contains the entries to be added to the relation (Any, Input)

Method:

None

Design Requirements:

1. Only integer, single-precision, or string attributes may be added with this routine.

Error Conditions:

None

Database Relational Utility Module: **READDM**

Entry Point: **READDM**

Purpose:

To add a new entry to a relation that contains double-precision, or mixed precision, attributes.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL READMM ( RELNAM, SNGL, DBLE )
```

RELNAM	Name of the relation (Character, Input)
SNGL	Array that contains the single-precision entry data (Any, Input)
DBLE	Array that contains the double-precision entry data (Double, Input)

Method:

None

Design Requirements:

None

Error Conditions:

None

Database Relational Utility Module: **RECLRC**

Entry Point: **RECLRC**

Purpose:

To clear the conditions set on a relational entity without performing a **DBCLOS**.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL RECLRC ( ENTNAM )
```

ENTNAM	The name of the relational entity (Character, Input)

Method:

None

Design Requirements:

None

Error Conditions:

None

Database Relational Utility Module: **RECOND**

Entry Point: **RECOND**

Purpose:

To define a condition, or constraint, for a relational attribute prior to performing a get operation (see also **RESETC**).

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL RECOND ( RELNAM, ATTRNAM, RELOP, VAL )
```

RELNAM	Name of the relation (Character, Input)
ATTRNAM	Name of the attribute (Character, Input)
RELOP	The relational operator for the constraint; one of **'EQ'**, **'NE'**, **'GT'**, **'LT'**, **'GE'**, **'LE'**, **'MAX'**, **'MIN'** (Character, Input)
VAL	The value to be tested (Any, Input)

Method:

None

Design Requirements:

1. **VAL** must be the same type as the **ATTRNAM**. All **RELOP**s are legal for attributes of type **'INT'**, **'KINT'**, **'RSP'**, and **'RDP'**. Only **'EQ'** and **'NE'** are valid for attribute types **'STR'** and **'KSTR'**. Attribute types of **'AINT'**, **'ARSP'**, and **'ARDP'** may not be used in a condition. Also, for attributes of type **'STR'** or **'KSTR'**, their length must be 8 or fewer characters. Note that string attribute values are passed as hollerith data.
2. Any **RECOND** call removes any existing conditions, that is, it performs an **RECLRC** internally.

Error Conditions:

None

Database Relational Utility Module: **RECPOS**

Entry Point: **RECPOS**

Purpose:

To position to a specific entry and return the row number, if present.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL RECPOS ( RELNAM, KEY, VAL, ROWNUM )
```

RELNAM	Name of the relation (Character, Input)
KEY	Name of a keyed attribute (Character, Input)
VAL	The desired value of the keyed attribute (Integer, Input)
ROWNUM	The row number satisfying the condition. If zero, no entries were found (Integer, Output)

Method:

None

Design Requirements:

1. If the **KEY** is a character attribute, it must be of length eight and **VAL** must contain the hollerith representation of the desired value. The conversion from character to hollerith must be made with the **DBMDCH** routine.

Error Conditions:

None

Database Relational Utility Module: **REENDC**

Entry Point: **REENDC**

Purpose:

To end the definition for a relational entity.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL REENDC
```

Method:

None

Design Requirements:

None

Error Conditions:

None

Database Relational Utility Module: **REGB**

Entry Point: **REGB**

Purpose:

To fetch all of the entries of the requested relation that satisfy the specified projection and constraints (see also **REGBM**).

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL REGB ( RELNAM, BUF, INUM, ISTAT )
```

RELNAM	Name of the relation (Character, Input)
BUF	Array that will contain the entries (Any, Output)
INUM	The number of entries fetched (Integer, Output)
ISTAT	Status return (Integer, Output) **0** entries successfully fetched **201** no entries found

Method:

None

Design Requirements:

1. Only integer or real, single-precision or string attributes may be fetched with this routine.

Error Conditions:

None

Database Relational Utility Module: **REGBM**

Entry Point: **REGBM**

Purpose:

To fetch all of the entries of the requested relation that satisfy the specified projection and constraints and that contain double-precision, or mixed precision attributes.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL REGBM ( RELNAM, SNGL, DBLE, INUM, ISTAT )
```

RELNAM	Name of the relation (Character, Input)
SNGL	Array that will contain single-precision entry data (Integer, Real Single-precision, or String, Output)
DBLE	Array that will contain double-precision entry data (Double, Output)
INUM	The number of entries fetched (Integer, Output)
ISTAT	Status return (Integer, Output) **0** entries successfully fetched **201** no entries found

Method:

None

Design Requirements:

None

Error Conditions:

None

Database Relational Utility Module: **REGET**

Entry Point: **REGET**

Purpose:

To fetch an entry of a relation that satisfies the given projection and constraint conditions.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL REGET ( RELNAM, BUF, ISTAT )
```

RELNAM	Name of the relation (Character, Input)
BUF	Array that will contain the entry data (Any, Output)
ISTAT	Status return (Integer, Output) **0** entries successfully fetched **201** no entries found

Method:

None

Design Requirements:

None

Error Conditions:

None

Database Relational Utility Module: **REGETM**

Entry Point: **REGETM**

Purpose:

To fetch an entry of a relation that satisfies the given projection and constraint conditions, and that contains double-precision, or mixed precision attributes.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL REGETM ( RELNAM, SNGL, DBLE, ISTAT )
```

RELNAM	Name of the relation (Character, Input)
SNGL	Array that will contain single-precision entry data (Integer, Real Single-precision, or String) (Any, Output)
DBLE	Array that will contain double-precision entry data (Double, Output)
ISTAT	Status return (Integer, Output) **0** entries successfully fetched **201** no entries found

Method:

None

Design Requirements:

None

Error Conditions:

None

Database Relational Utility Module: **RENULx**

Entry Points: **RENULD, RENULI, RENULR, RENULS**

Purpose:

To check if a double-precision, integer, real, or character attribute has a null value. Attributes excluded from the projection when a relational entry is added are given such null values.

MAPOL Calling Sequence:

None

Application Calling Sequences:

```
RENULD ( FIELDD )
RENULI ( FIELDI )
RENULR ( FIELDR )
RENULS ( FIELDS )
```

FIELDD	Double precision attribute value (Input)
FIELDI	Integer attribute value (Input)
FIELDR	Real, single-precision attribute value (Input)
FIELDS	String attribute value (Input)
RENULD **RENULI** **RENULR** **RENULS**	Logical values (output) **TRUE** if **FIELDx** is null

Method:

None

Design Requirements:

1. The string attribute, **FIELDS**, must be passed as a hollerith.

Error Conditions:

None

Database Relational Utility Module: **REPOS**

Entry Point: **REPOS**

Purpose:

To position a relation to an entry with a given keyed attribute.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL REPOS ( RELNAM, KEY, VAL )
```

RELNAM	Name of the relation (Character, Input)
KEY	Name of a keyed attribute (Character, Input). The special attribute name of **'ENTRYNUM'** with a **VAL=1** may be used to reposition an entity to the beginning.
VAL	The desired value of the keyed attribute (Integer, Input)

Method:

None

Design Requirements:

1. The attribute must be keyed.
2. If **KEY='ENTRYNUM'** then **VAL** must be 1.

Error Conditions:

1. A database fatal error occurs if the requested entry does not exist.

Database Relational Utility Module: **REPROJ**

Entry Point: **REPROJ**

Purpose:

To define the projection, or subset of attributes, for the relation prior to performing updates, adds or gets of entries.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL REPROJ ( RELNAM, NATTR, ATTRLIST )
```

RELNAM	Name of the relation (Character, Input)
NATTR	Number of attributes in the projection (Integer, Input)
ATTRLIST	Array containing the attribute names that define the projection (Character, Input)

Method:

None

Design Requirements:

None

Error Conditions:

None

Database Relational Utility Module: **REQURY**

Entry Point: **REQURY**

Purpose:

To retrieve the schema of a relation.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL REQURY ( RELNAM, NATTR, ATTRLIST, ATTRTYPE, ATTRLEN, TOTLEN )
```

RELNAM	Name of the relation (Character, Input)
NATTR	Number of attributes (Integer, Output)
ATTRLIST	Array containing the attribute names (Character, Output)
ATTRTYPE	Array containing the attribute types (Character, Output)

'INT'	Integer attribute
'KINT'	Keyed integer attribute
'AINT'	Array of integers
'RSP'	Real, single-precision attribute
'ARSP'	Array of single-precision
'RDP'	Real, double-precision attributes
'ARDP'	Array of double-precision
'STR'	String attribute
'KSTR'	Keyed string attribute

ATTRLEN	Array defining the number of elements in an array attribute or the number of characters in a string attribute (Integer, Output)
TOTLEN	Total length of schema in words (Integer, Output)

Method:

None

Design Requirements:

None

Error Conditions:

None

Database Relational Utility Module: **RESCHM**

Entry Point: **RESCHM**

Purpose:

To define the schema of a relation being created by a functional module.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL RESCHM ( RELNAM, NATTR, ATTRLIST, ATTRTYPE, ATTRLEN )
```

RELNAM	Name of the relation (Character, Input)
NATTR	Number of attributes (Integer, Input)
ATTRLIST	Array containing the attribute names (Character, Input)
ATTRTYPE	Array containing the attribute types (Character, Input)

'INT'	Integer attribute
'KINT'	Keyed integer attribute
'AINT'	Array of integers
'RSP'	Real, single-precision attribute
'ARSP'	Array of single-precision
'RDP'	Real, double-precision attributes
'ARDP'	Array of double-precision
'STR'	String attribute
'KSTR'	Keyed string attribute

ATTRLEN	Array defining the number of elements in an array attribute or the number of characters in a string attribute (Integer, Input)

Method:

None

Design Requirements:

1. The relation **RELNAM** must not already have a schema defined.
2. Keyed attributes must have uniform values for all rows.
3. No attributes may have the name **'ENTRYNUM'**
4. Attributes of the type **'KSTR'** must have length of 8 or fewer characters.

Error Conditions:

None

Database Relational Utility Module: **RESETC**

Entry Point: **RESETC**

Purpose:

To define additional conditions, or constraints, for a relational attribute prior to performing a get operation (see also **RECOND**).

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL RESETC ( BOOL, ATTERNAM, RELOP, VAL )
```

BOOL	The boolean operation **'OR'** or **'AND'** (Character, Input)
ATTRNAM	Name of the attribute (Character, Input)
RELOP	The relational operator for the constraint; one of **'EQ'**, **'NE'**, **'GT'**, **'LT'**, **'GE'**, **'LE'**, **'MAX'**, **'MIN'** (Character, Input)
VAL	The value to be tested (Any, Input)

Method:

None

Design Requirements:

1. **VAL** must be the same type as the **ATTRNAM**. A maximum of 10 conditions may be specified. All **RELOP**s are legal for attributes of type **'INT'**, **'KINT'**, **'RSP'**, and **'RDP'**. Only **'EQ'** and **'NE'** are valid for attribute types **'STR'** and **'KSTR'**. Attribute types of **'AINT'**, **'ARSP'**, and **'ARDP'** may not be used in a condition. Also, for attributes of type **'STR'** or **'KSTR'**, their length must be 8 or fewer characters. Only one condition may have a **'MAX'** or **'MIN'** **RELOP**. Note that string attribute values are passed as hollerith data.

Error Conditions:

None

Database Relational Utility Module: **RESORT**

Entry Point: **RESORT**

Purpose:

To sort a relation on one or more of its attributes.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL RESORT ( RELNAM, NATTR, SORTTYPE, ATTRLIST, KORE )
```

RELNAM	Name of the relation (Character, Input)
NATTR	The number of attributes to be sorted (Integer, Input)
SORTYPE	The type of sort for each attribute (Character, Input)
	'ASC' Ascending
	'DES' Descending
ATTRLIST	A list of the attributes to be sorted (Character, Input)
KORE	Base address of open core (Input)

Method:

None

Design Requirements:

1. The sort sequence is performed in the order that the attributes are specified in **ATTRLIST**.
2. The relation **RELNAM** must be closed when **RESORT** is called.

Error Conditions:

None

Database Relational Utility Module: **REUPD**

Entry Point: **REUPD**

Purpose:

To update the current relational entry.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL REUPD ( RELNAM, BUF )
```

RELNAM	Name of the relation (Character, Input)
BUF	Array that contains updated entry data (Any, Input)

Method:

None

Design Requirements:

1. Only integer, single-precision or string attributes may be updated with this routine.

Error Conditions:

None

Database Relational Utility Module: **REUPDM**

Entry Point: **REUPDM**

Purpose:

To update the current relational entry that contains double-precision, or mixed precision, attributes.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL REUPDM ( RELNAM, SNGL, DBLE )
```

RELNAM	Name of the relation (Character, Input)
SNGL	Array that contains the single-precision entry data (Any, Input)
DBLE	Array that contains the double-precision entry data (Double, Input)

Method:

None

Design Requirements:

None

Error Conditions:

None

8.6. UTILITIES FOR UNSTRUCTURED ENTITIES

Unstructured database entities are used primarily for scratch I/O by modules or for saving data that are generally used on an all-or-nothing basis. That is, random access to anything other than a complete record is not used. The utilities to support this type of data are shown below:

SUBROUTINE	FUNCTION
UNPOS	Positions to a given unstructured record
UNRPOS	
UNSTAT	Returns the length of a record
UNGET	Gets, or fetches, an entire record
UNGETP	Gets, or fetches, a partial record
UNPUT	Adds a new record to the unstructured entity
UNPUTP	Adds a partial record to the entity

8.6.1. Generating an Unstructured Entity

As seen in Subsection 8.2, the first step in generating any entity is to perform a **DBCREA**. This is followed by a **DBOPEN**, any desired I/O activity, and finally a **DBCLOS**. Suppose, for example, the local coordinates X, Y, and Z of 1000 grid points have been computed and are in a block of dynamic memory called **GRID** whose location pointer is **IGRD** (see Section 8.3). Further, assume that these coordinates have also been converted to the basic coordinate system, and that these transformed coordinates are located in block **NGRD** with pointer **IGND**. These data will be used in a subsequent routine or module in their entirety. It will therefore be written into an unstructured entity called **COORD** in two distinct records. The code segment to perform this is shown below:

```
C
C        CREATE THE NEW ENTITY AND OPEN FOR I/O
C
         CALL DBCREA ('COORD', 'IUN')
         CALL DBOPEN ('COORD', INFO, 'R/W', 'FLUSH', ISTAT)
C
C        WRITE THE TWO UNSTRUCTURED RECORDS
C
         CALL UNPUT ('COORD', Z (IGRD), 3000)
         CALL UNPUT ('COORD', Z (IGND), 3000)
C
C        I/O COMPLETE CLOSE ENTITY
C
         CALL DBCLOS ('COORD')
```

The **UNPUT** call loads a complete record into the entity. Therefore, the above operations generate two records in **COORD**. If an operation is being performed "on-the-fly," or complete records do not fit in memory, then a "partial" put, **UNPUTP**, may be performed.

Now assume that the local coordinates are all in memory, but that the transferred coordinates will be generated on a point-by-point basis and written to the **COORD** entity. Subroutine **TRANSF** transforms a set of three local coordinates to basic coordinates stored in a local array **XNEW**. This is illustrated below:

```
C
C         CREATE AND OPEN THE ENTITY
C
          CALL DBCREA ('COORD', 'UN')
          CALL DBOPEN ('COORD', INFO, 'R/W', 'FLUSH', ISTAT)
C
C         FIRST WRITE THE LOCAL COORDINATE
C
          CALL UNPUT ('COORD', Z (IGRD), 3000)
C
C         NEXT, COMPUTE NEW COORDINATES ONE-AT-A-TIME
C
          DO 100 I=0,2999,3
              CALL TRANSF (Z (IGRD+I), XNEW)
              CALL UNPUTP ('COORD', XNEW, 3)
100       CONTINUE
C
C         TERMINATE PARTIAL RECORD AND CLOSE
C
          CALL UNPUT ('COORD', 0,0)
          CALL DBCLOS ('COORD')
```

Note that a record of an unstructured entity that is created by partial puts must be "closed" by a call to **UNPUT**. In this case, the final put operation does not extend the record but only terminates it.

8.6.2. Accessing an Unstructured Entity.

The **UNPUT** and **UNPUTP** utilities have direct analogs in **UNGET** and **UNGETP** for the retrieval of data. Three other utilities are available for data access. The first two, **UNPOS** and **UNRPOS**, allow an unstructured entity to be positioned to a specific record. Note that this access is much faster if the entity was created with an index structure, that is, the **DBCREA** call specified type **'IUN'**. The third utility, **UNSTAT**, is used to find length of a given record. These utilities are demonstrated in the example below. the second record of **COORD** will be accessed and each coordinate set used individually. It is not assumed that the number of grid points is known to this application.

```
C
C         OPEN THE ENTITY, READONLY MODE
C
C
          CALL DBOPEN ('COORD', INFO, 'RO', 'NOFLUSH', ISTAT)
C
C         NOFLUSH PROTECTS AGAINST DESTROYING THE DATA,
C         NOW, POSITION TO RECORD 2, GET LENGTH OF RECORD
C
          CALL UNPOS ('COORD', 2)
          CALL UNSTAT ('COORD', IREC, LEN)
C
C         LEN IS THE NUMBER OF WORDS IN THE RECORD,
C         FETCH AND USE COORDINATES ONE-AT-A-TIME
C
          NGRID=LEN/3
          DO 100 I=1, NGRID
              CALL UNGETP ('COORD', XNEW, 3)
C
C         USE THE XNEW VALUES HERE
C
100       CONTINUE
C
C         I/O COMPLETE, CLOSE THE ENTITY
C
          CALL DBCLOS ('COORD')
```

8.6.3. Modifying an Unstructured Entity.

It is also possible to modify, or update, the contents of an individual record within an unstructured entity. The only limitation to this feature is that the length of the record must be the same as, or less than, the length of the originally created record.

Database Unstructured Utility Module: **UNGET**

Entry Point: **UNGET**

Purpose:

To fetch a complete record from an unstructured entity.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL UNGET ( NAME, ARAY, NWORD )
```

NAME	Name of the unstructured entity (Character, Input)
ARAY	Array that will contain the unstructured record (Integer, Output)
NWORD	The number of single-precision words to be transferred (Integer, Input)

Method:

If **NWORD** is less than the total number of words, the remaining data will not be retrieved. **UNGET** positions the entity to the next record after the retrieval.

Design Requirements:

None

Error Conditions:

None

Database Unstructured Utility Module: **UNGETP**

Entry Point: **UNGETP**

Purpose:

To fetch a portion of an unstructured record.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL UNGETP ( NAME, ARAY, NWORD )
```

NAME	Name of the unstructured entity (Character, Input)
ARAY	Array that will contain the unstructured record (Integer, Output)
NWORD	The number of single-precision words to be transferred (Integer, Input)

Method:

Following the retrieval, the entity is still positioned at the same record, a subsequent **UNGET** or **UNGETP** will get the next words in the record.

Design Requirements:

None

Error Conditions:

None

Database Unstructured Utility Module: **UNPOS**

Entry Point: **UNPOS**

Purpose:

To position an unstructured entity to a specific record.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL UNPOS ( NAME, RECNO )
```

NAME	Name of the unstructured entity (Character, Input)
RECNO	Record number (Integer, Input)

Method:

None

Design Requirements:

None

Error Conditions:

None

Database Unstructured Utility Module: **UNPUT**

Entry Point: **UNPUT**

Purpose:

To add a record to an unstructured entity. The record is terminated after the transfer.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL UNPUT ( NAME, ARAY, NWORD )
```

NAME	Name of the unstructured entity (Character, Input)
ARAY	Array containing the record to be added (Any, Input)
NWORD	The number of words to be transferred (Integer, Input)

Method:

None

Design Requirements:

None

Error Conditions:

None

Database Unstructured Utility Module: **UNPUTP**

Entry Point: **UNPUTP**

Purpose:

To add a partial record to an unstructured entity. The record is not terminated after the transfer.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL UNPUTP ( NAME, ARAY, NWORD )
```

NAME	Name of the unstructured entity (Character, Input)
ARAY	Array containing the record to be added (Any, Input)
NWORD	The number of words to be transferred (Integer, Input)

Method:

None

Design Requirements:

None

Error Conditions:

None

Database Unstructured Utility Module: **UNRPOS**

Entry Point: **UNRPOS**

Purpose:

To position an unstructured entity to a specific record defined as an increment from the current record.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL UNSTAT ( NAME, DELRID )
```

NAME	Name of the unstructured entity (Character, Input)
DELRID	Record number increment relative to the current position (Integer, Input)

Method:

None

Design Requirements:

None

Error Conditions:

None

Database Unstructured Utility Module: **UNSTAT**

Entry Point: **UNSTAT**

Purpose:

To return the length, in single-precision words, of the current record.

MAPOL Calling Sequence:

None

Application Calling Sequence:

```
CALL UNSTAT ( NAME, RECNO, LEN )
```

NAME	Name of the unstructured entity (Character, Input)
RECNO	Current record number (Integer, Output)
LEN	Record length in single-precision words (Integer, Output)

Method:

None

Design Requirements:

None

Error Conditions:

None

THIS PAGE LEFT INTENTIONALLY BLANK

9. DATABASE ENTITY DESCRIPTIONS

All intermodular communication in ASTROS is done through data stored on the CADDB database. As a result, there are many relations, matrices, and unstructured entities that are defined to store the data requisite to the analyses. This section provides a description of each of the database entities that are used in the ASTROS system. These data are useful both to the ASTROS programmer, who needs to know the description of each of the entities to interpret and modify the ASTROS source code, and to the general user in that these data are available on the database for other uses such as the Interactive CADDB Environment, ICE. For supplementary post-processing, the matrix entities may be combined using the MAPOL language to generate data not otherwise computed. At a more sophisticated level, a user-written Fortran module may take existing data from the entities to perform more advanced operations that are beyond the capabilities of the ASTROS executive system.

The entities are documented in alphabetical order and every entity which is used in intermodule communication is included. Those entities which are used for scratch storage within a module are documented in-line rather than in the Programmer's Manual. The entities presented here fall into three categories: system level, hidden entities, and MAPOL entities. The first include those entities that communicate system information between modules. The hidden and MAPOL entities are those which are declared in the MAPOL sequence itself. The hidden entities do not subsequently appear in the MAPOL sequence; their declaration is included as a convenience to the ASTROS executive system. The most common example of a hidden entity is any relation associated with a Bulk Data entry. These relations are used internally by numerous modules but they do not appear in the MAPOL calling sequences because their inclusion would result in an impractically large number of arguments. MAPOL entities are the most relevant and include most matrix entities and a large number of relational entities that are used to pass data between engineering modules.

The entity documentation format is slightly different for each of the entity classes. The core information, however, is the same for each class and includes the entity name, a description of its contents, the modules that create or add data to the entity, and any additional notes required to define special data handling functions. Each entity class then has an additional set of information.

Matrix entities have a section labeled *Matrix Form* which gives the row and column dimensions of the matrix and indicates the numeric precision and the form of the entity. Relational entities have a section called *Relation Attributes* which lists the schema of the relation and defines the meaning of each of the attributes. Finally, unstructured entities have have a section labeled *Entity Structure* which lists and defines the number and contents of the records of the entity.

Entity: **AA**

Entity Type: Matrix

Description: Acceleration in the a-set merged from the AL and AR matrices (see AG).

Entity: **AAICMAT**

Entity Type: Subscripted Matrix

Description: Aerodynamic influence coefficient matrix for an antisymmetric boundary condition and a given Mach number. The Mach number associated with a given subscript is given in the TRIM relation.

Matrix Form: Square, real and asymmetric. The dimension of the matrix is equal to the number of panels in the steady aerodynamics USSAERO model.

Created By: Module STEADY

Notes:

1. STEADY creates as many matrices as there are distinct antisymmetric Mach numbers in the user's input packet. If a combination of symmetric and antisymmetric Mach numbers are used, the MINDEX changes for each distinct Mach number. An AAICMAT entity is created for a given MINDEX only if the corresponding Mach number requires the antisymmetric boundary conditions. It is possible, therefore, that, in the range from 1 to MINDEX, some subscript values will not have a corresponding AAICMAT.

Entity: **ACPT**

Entity Type: Unstructured

Description: Contains one record for each independent group of aerodynamic elements with data needed to generate the aerodynamic matrices.

Entity Structure:

RECORD	WORD	TYPE	ITEM
1	1	I	Key word, 1 for doublet lattice
	2	I	Number of panels, NP
	3	I	Number of strips, NSTRIP
	4	I	Number of boxes, NTP
	5	R	F, fraction of box chord from center of pressure to downwash center
	NP WORDS	I	NCARAY, boxes per chord
	NP WORDS	I	NBARAY, last box on panel
	NSTRIP WORDS	R	YS aero coordinates of strip
	NSTRIP WORDS	R	ZS center
	NSTRIP WORDS	R	EE strip half width
	NSTRIP WORDS	R	SG sine of dihedral angle
	NSTRIP WORDS	R	CG cosine of dihedral angle
	NTP WORDS	R	XIC coordinate of center of pressure
	NTP WORDS	R	DELX box chord
	NTP WORDS	R	XLAM tangent of sweepback angle
	NTP WORDS	R	TR box taper ratios
2	1	I	Key word, 2 for Doublet Lattice with Bodies
	2	I	NJ, Number of J points
	3	I	NK, Number of K points
	4	I	NP, Number of Panels
	5	I	NB, Number of Bodies
	6	I	NTP, Number of Boxes
	7	I	NBZ, Number of Z Bodies
	8	I	NBY, Number of Y Bodies

RECORD	WORD	TYPE	ITEM
2 (cont)	9	I	NTZ, Number of Z Interference Body Elements
	10	I	NTY, Number of Y Interference Body Elements
	11	I	NTO, Sum of NTP + NTZ + NTY
	12	I	NTZS, Number of Z Slender Body Elements
	13	I	NTYS, Number of Y Slender Body Elements
	14	I	NSTRIP, Number of strips on panels
	NP WORDS	I	NCARAY, Boxes per chord
	NP WORDS	I	NBARAY, Last box on panel
	NP WORDS	I	NAS, Associated bodies per panel
	NB WORDS	I	*NBEA1, Number of interference elements
	NB WORDS	I	*NBEA2, Z-Y flag
	NB WORDS	I	*NSBEA, Number of slender elements
	NB WORDS	R	ZB, Z Body center
	NB WORDS	R	YB, Y Body center
	NB WORDS	R	AVR, Half-width of body
	NB WORDS	R	ARB, Cross-section aspect ratio
	NB WORDS	I	NFL, θ=distribution per body
	NB WORDS	R	XLE, X-leading edge
	NB WORDS	R	XTE, X-trailing edge
	NB WORDS	I	NT121, number θ1's for bodies
	NB WORDS	I	NT122, number θ2's for bodies
	NB+STRIP	R	ZS, Z - of strip center
	NB+STRIP	R	YS, Y - of strip center
	NSTRIP WORDS	R	EE, strip half-width
	NSTRIP WORDS	R	SG, sine of dihedral angle
	NSTRIP WORDS	R	CG, cosine of dihedral angle
	NTP+ ΣNBEA1	R	X, 3/4 chord
	NTP+ ΣNBEA1	R	DELX, box chord
	NTP WORDS	R	XIC, coordinates of center of pressure

RECORD	WORD	TYPE	ITEM
2 (cont)	NTP WORDS	R	XLAM, tangent of sweepback angle
	ΣNSBEA WORDS	R	AO, half-widths for bodies
	ΣNSBEA WORDS	R	XIS1, X - of slender leading edge
	ΣNSBEA WORDS	R	XIS2, X - of slender trailing edge
	ΣNSBEA WORDS	R	AOP, X-derivatives of body half-width
	ΣNBEA1 WORDS	R	RIA, Radius of interference elements
	ΣNAS WORDS	I	NASB, associated bodies
	ΣNFL WORDS	I	IFLA1, body with θ1 distribution
	ΣNFL WORDS	I	IFLA2, Body with θ2 distribution
	ΣNT121 WORDS	R	TN1A, θ1's for bodies
	ΣNT122 WORDS	R	TN2A, θ2's for bodies
*Sum of entries is noted by NBEAi, where i = 1, 2			

Entity: **AECOMPS**

Entity Type: Relation

Description: Contains data on the aerodynamic components in the planar and nonplanar steady aerodynamics model.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
MODEL	Integer	Planar or nonplanar steady aerodynamics model identifier = 1 for planar model = –1 for nonplanar model
ACID	Integer	Component identification number
MACROTYPE	Text (8)	Type of macroelement (CAERO6 or PAERO6)
GROUP	Integer	Group identification number
ACMPNT	Text (8)	Component type One of WING, FIN, CANARD, POD, or FUSEL
TYPE	Integer > 0	Type of degree of freedom. For STEADY aero models all DOF's are TYPE=1. Refer to AECOMPU for other types.
FIINTID	Integer	First internal degree of freedom on the macroelement
NCBOX	Integer	Number of chordwise boxes for lifting surfaces or number of circumferential boxes for bodies
NSBOX	Integer	Number of spanwise boxes for lifting surfaces or number of axial boxes for bodies
BNDRY	R vector (12)	Coordinates of the component corners in basic coordinates

Created By: Module STEADY

Notes:

1. The boundary coordinates are the x, y, z coordinates for each of the corners of lifting elements. Body elements do not use BNDRY.

   The data are in the following order:

   (1-3) Leading Edge Root
   (4-6) Trailing Edge Root
   (7-9) Trailing Edge Tip
   (10-12) Leading Edge Tip

Entity: **AECOMPU**

Entity Type: Relation

Description: Contains data on the aerodynamic components in the unsteady aerodynamics model.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
ACID	Integer	Component identification number
MACROTYPE	Text (8)	Type of macroelement, CAERO1 or CAERO2
GROUP	Integer	Group identification number
ACMPNT	Text (8)	Component type One of WING or BODY
TYPE	Integer > 0	Degree of freedom type =2 for WING and Z body elements =3 for Y body elements =4 for ZY body elements
FIINTID	Integer	First internal degree of freedom on the macroelement
NCBOX	Integer	Number of chordwise boxes for lifting surfaces or number of circumferential boxes for bodies
NSBOX	Integer	Number of spanwise boxes for lifting surfaces or number of axial boxes for bodies
BNDRY	R vector (12)	Coordinates of the component corners in basic coordinates

Created By: Module UNSTEADY

Notes:

1. The boundary coordinates are the x, y, z coordinates for each of the corners of lifting elements. Body elements do not use BNDRY.

   The data are in the following order:

   (A) Leading Edge Root
   (B) Trailing Edge Root
   (C) Trailing Edge Tip
   (D) Leading Edge Tip

Entity: **AEFACT**

Entity Type: Relation

Description: Contains aerodynamic input data as defined on the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
VALUE	Real	Data value

Created By: Module IFP

Note:

1. This relation contains one tuple for each value in each set defined on the AEFACT card entry.

Entity: **AERO**

Entity Type: Relation

Description: Contains basic aerodynamic data for use in unsteady aerodynamics as input from the bulk data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
ACSID	Integer ≥ 0	Coordinate system identification number for the aerodynamic coordinate system
REFC	Real > 0.0	Reference length for reduced frequency
RHOREF	Real > 0.0	Reference density

Created By: Module IFP

Entity: **AEROGEOM**

Entity Type: Relation

Description: Contains the aerodynamic planform geometric grid points for the planar and nonplanar steady aerodynamics model. These grid points are not used for data recovery, but can be used in combination with the "elements" in CAROGEOM to create an ASTROS FE model using RODs and QUADs that represents the paneling of the aero model.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
MODEL	Integer	=1 for the planar model =– 1 for the nonplanar model
GRIDID	Integer	Aerodynamic grid identification number
X	Real	Basic coordinates of the geometric point
Y	Real	
Z	Real	

Created By: STEADY and/or STEADYNP modules

Notes:

1. These grid points represent the airfoil and panel geometry of the aerodynamic model identified by the MODEL attribute. The connectivity of these grid points is given in the CAROGEOM entity.

Entity: **AEROS**

Entity Type: Relation

Description: Contains the basic parameters for static aeroelasticity as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
ACSID	Integer > 0	Aerodynamic coordinate system identification
RCSID	Integer > 0	Reference coordinate system for rigid body motions
REFC	Real > 0.0	Reference chord length
REFB	Real > 0.0	Reference span
REFS	Real > 0.0	Reference wing area
GREF	Integer > 0	Reference grid point
REFD	Real > 0.0	Body component reference diameter
REFL	Real > 0.0	Body component reference length

Created by: Module IFP

Entity: **AERUGEOM**

Entity Type: Relation

Description: Contains the aerodynamic planform geometric grid points for the flutter analysis model. These grid points are not used for data recovery, but can be used in combination with the "elements" in CAROUGEO to create an ASTROS FE model using ROD's and QUAD's that represents the paneling of the aero model.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
GRIDID	Integer	Aerodynamic grid identification number
X	Real	Basic coordinates of the geometric point
Y	Real	
Z	Real	

Created By: UNSTEADY module

Notes:

1. The connectivity of these grid points is given in the CAROUGEO entity.

Entity: **AESURF**

Entity Type: Relation

Description: Contains the specification of an aerodynamic control surface as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
LABEL	Text (8)	Alphanumeric data identifying the control surface
TYPE	Text (8)	Surface type
ACID	Integer > 0	Aerodynamic component identification number for control surface definition
CID	Integer > 0	Coordinate system defining the surface hinge line
FBOX1	Integer > 0	First aerodynamic box on the control surface
LBOX1	Integer > 0	Last aerodynamic box on the control surface

Created By: Module IFP

Entity: **AF**

Entity Type: Matrix

Description: Merged from the AA matrix (see AG).

Entity: **AG**

Entity Type: Subscripted Matrix

Description: Contains the accelerations of the structural degrees of freedom.

Matrix Form: A variable sized matrix having one row for each structural degree of freedom and one column for each load condition in the current boundary condition.

Created By: MAPOL

Notes:

1. The dimension of this subscripted matrix must be large enough for all optimization and analysis boundary conditions.
2. This entity is only filled for analysis of unrestrained structures.
3. The MAPOL sequence recovers this matrix in the following order (see the Theoretical Manual for the explicity form of this recovery):

$$\begin{bmatrix} AL \\ AR \end{bmatrix} \rightarrow AA$$

$$\begin{bmatrix} \varphi \\ AA \end{bmatrix} \rightarrow AF$$

$$\begin{bmatrix} \varphi \\ AF \end{bmatrix} \rightarrow AN$$

$$\begin{bmatrix} UM^* \\ AN \end{bmatrix} \rightarrow AG$$

*UM contains accelerations in the M-set. The entity name is reused in the MAPOL sequence.

Entity: **AGA**

Entity Type: Matrix

Description: Contains the active acceleration vectors for the current boundary condition.

Matrix Form: A matrix having one column for each active acceleration vector and one row for each degree of freedom in the structural model.

Created By: MAPOL

Notes:

1. This entity is only generated during sensitivity evaluation of unrestrained boundary conditions.

Entity: **AICMAT**

Entity Type: Subscripted Matrix

Description: Aerodynamic influence coefficient matrix for a symmetric boundary condition and a given Mach number. The Mach number associated with a given subscript is given in the TRIM relation.

Matrix Form: Square, real and asymmetric. The dimension of the matrix is equal to the number of panels in the steady aerodynamics USSAERO model.

Created By: Module STEADY

Notes:

1. STEADY creates as many matrices as there are distinct symmetric Mach numbers in the user's input packet. If a combination of symmetric and anti-symmetric Mach numbers are used, the MINDEX changes for each distinct Mach number. An AICMAT entity is created for a given MINDEX only if the corresponding Mach number requires the symmetric boundary conditions. It is possible, therefore, that, in the range from 1 to MINDEX, some subscript values will not have a corresponding AICMAT.

Entity: **AICS**

Entity Type: Matrix

Description: Steady aerodynamic influence coefficient matrix for a given boundary condition in the structural coordinates.

Matrix Form: Square, real and asymmetric. The dimension of the matrix is equal to the number of degrees of freedom in the f-set.

Created By: MAPOL

Notes:

1. This matrix is derived from splining the AICMAT or AAICMAT matrix to the structural degrees of freedom.

Entity: **AIRFOIL**

Entity Type: Relation

Description: Contains the airfoil properties to be used in the aerodynamic analyses as defined on the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
ACID	Integer > 0	Aircraft component identification number
ACMPNT	Text (8)	Component type (i.e. WING)
CP	Integer > 0	Coordinate system identification number
CHORD	Integer > 0	AEFACT setid for the chordwise division points
UST	Integer > 0	AEFACT setid for the upper surface half thicknesses
LST	Integer ≥ 0	AEFACT setid for the lower surface half thicknesses
CAMBER	Integer ≥ 0	AEFACT setid for the camber ordinants
RADIUS	Real > 0	Airfoil leading edge radius
X1, Y1, Z1	Real	Location of point 1 in coordinate system CP
X12	Real > 0.0	Edge chord length in coordinate system CP
IPANEL	Integer ≥ 0	AEFACT setid containing chordwise cuts for wing paneling

Created By: IFP Module

Entity: **AIRFRC**

Entity Type: Subscripted Matrix

Description: Rigid body aerodynamic load vectors for a given Mach number. There is one vector for each configuration parameter associated with the Mach index. There are six symmetric parameters: NX, NZ, QACCEL, THKCAM, ALPHA and QRATE and 6 antisymmetric parameters: NY, PACCEL, RACCEL, BETA, PRATE, RRATE. In addition, each symmetric and antisymmetric control surface AESURF will generate a column. For a given subscript, the AIRFRC matrix contains the six columns for the symmetric parameters plus one column for each symmetric AESURF if the SYMMETRIC forces are needed for the associated Mach number. It contains six columns for the antisymmetric parameters and one column for each antisymmetric AESURF if the ANTISYMMETRIC forces are needed for the associated Mach number. If both are needed, all 12 parameters and all AESURF entries in the model have columns. The ordering of the columns corresponds to the order of entries in the STABCF entity.

Matrix Form: Rectangular and real. The number of rows is equal to the number of panels in the steady aerodynamics model while the number of columns is equal to the number of entries in the STABCF relation that have the same MACHINDX value as the subscript value. The columns of AIRFRC are stored in the same order as the entries in STABCF. Refer to the STABCF entity for more details.

Created By: STEADY Module

Notes:

1. STEADY creates as many matrices as there are distinct Mach numbers in the bulk data packet.

Entity: **AJJTL**

Entity Type: Matrix

Description: List of unsteady aerodynamic matrices to compute panel pressures due to slopes at the control point.

Matrix Form: Square, complex matrix with the number of rows and colums equal to the number of aerodynamic panels.

Created By: AMP

Notes:

1. AJJTL is a matrix list with the number of matrices equal to the number of M-k pairs in the input stream.

Entity: **AL**

Entity Type: Matrix

Description: Acceleration in the l-set obtained from D and AR (see AG).

Entity: **AMAT**

Entity Type: Matrix

Description: Matrix containing the sensitivity of the constraints to changes in the design variables.

Matrix Form: The number of columns is equal to the number of active constraints. The number of rows is equal to the number of design variables.

Created By: Modules ACTCON, AEROEFFS, AEROSENS, FREQSENS, FLUTSENS, MAKDFV, MKAMAT, and LAMINSNS

Notes:

1. The columns are written in the order they appear on the CONST relation except that, for a given boundary condition, all the constraints for a given subcase are grouped together. On the CONST relation, these constraints are grouped by type.
2. See CONST.
3. CONST and AMAT are brought into alignment in DESIGN where the CONST tuples are ordered to have all subcases grouped together.

Entity: **AR**

Entity Type: Matrix

Description: Contains the accelerations for the support degrees of freedom (see AG).

Matrix Form: A variable sized matrix having one row for each support degree of freedom and one column for each load condition in the current boundary condition.

Created By: Module INERTIA

Notes:

1. This matrix is only formed for the analysis of unrestrained structures.

Entity: **ASET**

Entity Type: Relation

Description: Contains the external grid identification numbers and components associated with the analysis set as defined on the ASET entries of the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	ASET identification number
GRID1	Integer > 0	Grid or scalar point identification number
COMPNTS	Integer ≥ 0	Component number; Zero for scalar points, 1-6 for grid points

Created By: Module IFP

Notes:

1. Used by the MKUSET module to build the USET relation.

Entity: **ASET1**

Entity Type: Relation

Description: Contains the external grid identification numbers and components associated with the analysis set as defined on the ASET1 entries of the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	ASET identification number
COMPNTS	Integer ≥ 0	Component number; Zero for scalar points, 1-6 for grid points
GRID1	Integer > 0	Grid or scalar point identification number

Created By: Module IFP

Notes:

1. Used by the MKUSET module to build the USET relation.

Entity: **ATTACH**

Entity Type: Relation

Description: Contains the definitions of aerodynamic boxes whose forces are to be attached to a referenced grid as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0	Element identification number
MACROID	Integer > 0	Element identification number of an aerodynamic macroelement
ID1	Integer > 0	External box identification number of the first aero box on the macroelement
ID2	Integer > 0	External box identification number of the last aero box on the macroelement
REFGRD	Integer > 0	The external identification number of the referenced grid point

Created By: Module IFP

Entity: **AXSTA**

Entity Type: Relation

Description: Contains the body axial station parameters for the aerodynamic model as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
BCID	Integer > 0	Body component identification number
XSTA	Real	X ordinate of body station
CBOD	Real	Z ordinate of body station
ABOD	Real > 0.0	Body cross-sectional area at XBOD
YRAD	Integer ≥ 0	AEFACT setid containing the y-ordinates of the body section
ZRAD	Integer ≥ 0	AEFACT setid containing the Z-ordinates of the body section

Created By: Module IFP

Entity: **BDD**

Entity Type: Matrix

Description: Damping matrix in the direct dynamic set.

Matrix Form: Square matrix with the number of rows and columns equal to the number of degrees of freedom in the d-set.

Created By: DMA

Entity: **BEAMEST**

Entity Type: Relation

Description: Contains the element summary data for the BAR element.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
PID	Integer > 0	Element property identification number
PTYPE	Text(8)	Element property type
SIL1	Integer > 0	Internal grid point id for end A
SIL2	Integer > 0	Internal grid point id for end B
ORIENTX	Real	Orientation vector for element
ORIENTY	Real	
ORIENTZ	Real	
ICSSV	Integer ≥ 0	The external coordinate system in which the orientation vector is defined.
PINA, PINB	Integer ≥ 0	The offset pinned degrees of freedom for ends A and B
OFFSETAX	Real	The offset vectors for ends A and B
OFFSETAY	Real	
OFFSETAZ	Real	
OFFSETBX	Real	
OFFSETBY	Real	
OFFSETBZ	Real	
MID1	Integer > 0	The material id for the element
AREA	Real > 0	The beam cross-sectional area
I1	Real ≥ 0	The area moment of inertia (Plane 1)
I2	Real ≥ 0	The area moment of inertia (Plane 2)
TORSION	Real ≥ 0	The beam torsional constant
NSM	Real ≥ 0	The beam non structural mass
C1, C2, D1, D2	Real	Element stress recovery coefficients
E1, E2, F1, F2	Real	
KFACT1	Real	Shear area factor (plane 1)

NAME	TYPE/KEY	DESCRIPTION
KFACT2	Real	Shear area factor (plane 2)
I12	Real	Beam product of inertia
R1SQR	Real	Inertia term; Definition for design
R2SQR	Real	
ALPHA	Real	
COORD1	Integer ≥ 0	External coordinate system of end A
X1, Y1, Z1	Real	Basic coordinates of end A
COORD2	Integer ≥ 0	External coordinate system of end B
X2, Y2, Z2	Real	Basic coordinates of end B
SCON	Integer	Stress constraint flag
DESIGN	Integer	Design flag
STHRM	Real	Thermal stress term
STHRMA	Real	Thermal strain term
TREFPT	Integer	Pointer to TREF entity for thermal stress/load evaluation
NLFLAG	Integer ≥ 0	Design variable nonlinear flag = 0 Design variable is linear = 1 Designed element using PBAR1 property, design variable is nonlinear
SHAPE	Text(8)	Cross section shape
NDSEC	Integer ≥ 0	Number of dimensions
DSEC1	Real ≥ 0.0	Cross section dimension 1
D1DES1	Integer ≥ 0	Design flag for dimension 1 = 1 Designed dimension = 0 Non-designed demension
DSEC2	Real ≥ 0.0	Cross section dimension 2
D2DES	Integer ≥ 0	Design flag for dimension 2 = 1 Designed dimension = 0 Non-designed demension
DSEC3	Real ≥ 0.0	Cross section dimension 3
D3DES	Integer ≥ 0	Design flag for dimension 3 = 1 Designed dimension = 0 Non-designed demension
DSEC4	Real ≥ 0.0	Cross section dimension 4

NAME	TYPE/KEY	DESCRIPTION
D4DES	Integer ≥ 0	Design flag for dimension 4 = 1 Designed dimension = 0 Non-designed demension
DSEC5	Real ≥ 0.0	Cross section dimension 5
D5DES	Integer ≥ 0	Design flag for dimension 5 = 1 Designed dimension = 0 Non-designed demension
DSEC6	Real ≥ 0.0	Cross section dimension 6
D6DES	Integer ≥ 0	Design flag for dimension 6 = 1 Designed dimension = 0 Non-designed demension
DSEC7	Real ≥ 0.0	Cross section dimension 7
D7DES	Integer ≥ 0	Design flag for dimension 7 = 1 Designed dimension = 0 Non-designed demension
DSEC8	Real ≥ 0.0	Cross section dimension 8
D8DES	Integer ≥ 0	Design flag for dimension 8 = 1 Designed dimension = 0 Non-designed demension
DSEC9	Real ≥ 0.0	Cross section dimension 9
D9DES	Integer ≥ 0	Design flag for dimension 9 = 1 Designed dimension = 0 Non-designed demension
DSEC10	Real ≥ 0.0	Cross section dimension 10
D10DES	Integer ≥ 0	Design flag for dimension 10 = 1 Designed dimension = 0 Non-designed demension
ELRSPREQ	Integer	User function element response flag = 1 Element response required = 0 Element response not required

Created by: Module MAKEST

Notes:

1. This relation contains one tuple for each beam element in the problem. It is built from the CBAR, PBAR and associated material and design relations.

Entity: **BFRC**

Entity Type: Matrix

Description: Matrix of rigid body load vectors for unit values of angle of attack, pitch rate and trim surface deflection.

Matrix Form: Rectangular real matrix with three columns and rows equal to the number of panels in the unsteady aerodynamics model.

Created By: Module BLASTFIT

Entity: **BGPDT**

Entity Type: Relation

Description: Contains the coordinates of the grid points in the basic coordinate system.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EXTID	Integer > 0	The grid or scalar point external identification number
INTID	Integer > 0	Equivalent internal identification number
FLAG	Integer > 0	Flag indicating the point is a grid point or a scalar point
CD	Integer	The displacement coordinate system for the grid point
X, Y, Z	Real	Spatial coordinates of the point in the basic coordinate system

Created by: Module BCBGPDT

Notes:

1. This relation contains one tuple for each grid or scalar point in the problem.
2. This relation is built from the GRID, SPOINT, EPOINT, CSTM and SEQGP relations.
3. The FLAG equals 6 if the point is a grid point and equals 1 if a scalar point and 0 if not in the g-set.
4. The internal identification number is determined by assigning INTID in increasing order of EXTID's.
5. Scalar points are also denoted by CD=-1; X=Y=Z=0.0.

Entity:	**BHH**
Entity Type:	Matrix
Description:	Damping matrix in the modal dynamic set.
Matrix Form:	Square matrix with the number of rows and columns equal to the number of degrees of freedom in the h-set.
Created By:	DMA
Notes:	

1. Info (11) for the entity contains a coupled flag
   = 0 Uncoupled
   = 1 Coupled
2. Info (12) contains damping data
   = 0 Modal damping only
   = 1 Viscous damping only
   = 2 Both modal and viscous damping

Entity: **BLAST**

Entity Type: Relation

Description: Contains the definition of parameters for use in nuclear blast response analysis as input from the bulk data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
BLID	Integer > 0	Blast set ID
ALT	Real > 0.0	Aircraft altitude
VEL	Real > 0.0	Aircraft velocity
WKT	Real > 0.0	Weapon yield
HGRD	Real	Height of the ground
KGRD	Integer	Flag denoting presence of ground
BALT	Real	Blast altitude
MACH	Real	Mach number
DELTAX	Real	X-distance from A/C to blast
DELTAY	Real	Y-distance from A/C to blast
TMIN	Real > 0.0	Minimum time used in curve fit
TMAX	Real > 0.0	Maximum time used in curve fit
NTIME	Integer > 0	Number of time steps
BMIN	Real > 0	Minimum decay value used in curve fit
BMAX	Real > 0	Maximum decay value used in curve fit
NBETA	Integer > 0	Number of decay values
SYMXY	Integer	Symmetry flag for xy plane
XYMXZ	Integer	Symmetry flag for xz plane
TRSURF	Text (8)	Trim surface label
NZ	Real	Load factor for trim calculation

Created By: Module IFP

Notes:

Entity: **BLGTJA**

Entity Type: Matrix

Description: A partition of matrix UGTKA used in the nuclear blast analysis to spline aerodynamic forces to the structure.

Matrix Form: Rectangular real matrix having one row for each structural degree of freedom in the a-set and one column for each panel in the unsteady aerodynamics model.

Created By: Module BLASTFIT

Entity: **BLSTJA**

Entity Type: Matrix

Description: A partition of matrix UGTKA used in the nuclear blast analysis to spline structural displacements to the panel slopes.

Matrix Form: Rectangular real matrix having one row for each structural degree of freedom in the a-set and one column for each panel in the unsteady aerodynamics model.

Created By: Module BLASTFIT

Entity: **BODY**

Entity Type: Relation

Description: Contains the body configuration parameters for the aerodynamic model as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
BCID	Integer > 0, key	Body component id
ACMPNT	Text (8)	Component type (i.e., POD)
CP	Integer ≥ 0	Coordinate system id for geometry input
NRAD	Integer ≥ 0	Number of equal body cuts used to define the body panels
X, Y, Z	Real	Ordinates of the body in coordinate system CP

Created By: Module IFP

Entity:	**BTEM**
Entity Type:	Matrix
Description:	A scratch matrix in the blast calculation.
Matrix Form:	Rectangular real matrix with the number of rows equal to the number of elastic modes retained in the blast analysis and three columns.
Created By:	MAPOL

Notes:

1. This matrix is the solution to the equation:

$$[\ \textbf{\textit{KEQE}}\ ]\ *\ [\ \textbf{\textit{BTEM}}\ ]\ =\ [\ \textbf{\textit{GFE}}\ ]$$

Entity: **CAERO1**

Entity Type: Relation

Description: Contains an aerodynamic macroelement (panel) in terms of two leading-edge locations and side chords. This is used for unsteady aerodynamics.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0	Element identification number
PID	Integer > 0	Identification number of property card. Used to specify associated bodies
CP	Integer > 0	Coordinate system for locating points 1 and 4
NSPAN	Integer ≥ 0	Number of spanwise boxes; if a positive value is given NSPAN, equal divisions are assumed; if zero or blank, a list of division points is given at LSPAN
NCHORD	Integer ≥ 0	Number of chordwise boxes; if a positive value is given NCHORD, equal divisions are assumed; if zero or blank, a list of division points is given at LCHORD
LSPAN	Integer ≥ 0	ID of an AEFACT data card containing a list of division points for spanwise boxes. Used only if NSPAN is zero or blank
LCHORD	Integer ≥ 0	ID of an AEFACT data card containing a list of division points for chordwise boxes. Used only if NCORD is zero or blank
IGID	Integer > 0	Interference group identification (aerodynamic elements with different IGID's are uncoupled)
X1, Y1, Z1	Real	Location of point 1 in coordinate system CP
X12	Real ≥ 0	Edge chord length (in aerodynamic coordinate system) (Cannot be zero if X43 is zero)
X4, Y4, Z4	Real	Location of point 4 in coordinate system CP
X43	Real ≥ 0	Edge chord length (in aerodynamic coordinate system)(Cannot be zero if X12 is zero)

Created By: Module IFP

Entity: **CAERO2**

Entity Type: Relation

Description: Contains the definition of an aerodynamic body for unsteady aerodynamics as input from the bulk data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0	Element identification number
PID	Integer > 0	Property identification number
CP	Integer ≥ 0	Coordinate system for locating point 1
NSB	Integer ≥ 0	Number of slender body elements
NINT	Integer ≥ 0	Number of interference elements
LSB	Integer ≥ 0	AEFACT identification number defining slender body division points
LINT	Integer ≥ 0	AEFACT identification number defining interference element division points
IGID	Integer > 0	Interference group identification
X1, Y1, Z1	Real	Location of point 1 in coordinate system CP
X12	Real > 0.0	Length of the body in the x-axis of the aerodynamic coordinate system

Created By: Module IFP

Entity: **CAERO6**

Entity Type: Relation

Description: Contains the definition of an aerodynamic macroelement used in aerodynamic analyses as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
ACID	Integer > 0	Aircraft component identification number
ACMPNT	Text (8)	Component type (i.e., WING)
CP	Integer ≥ 0	Coordinate system identification number for geometry input
GROUP	Integer > 0	Group identification number
SPAN	Integer ≥ 0	AEFACT setid for the division points of spanwise boxes
CHORD	Integer ≥ 0	AEFACT setid for the chordwise division points

Created By: Module IFP

Entity: **CAROGEOM**

Entity Type: Relation

Description: Contains the connectivity data for the aerodynamic planform of the planar and nonplanar steady aerodynamics model. These elements are not used for data recovery, but can be used in combination with the "grids" in AEROGEOM to create an ASTROS FE model using RODs and QUADs that represents the paneling of the aero model.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
MODEL	Integer	= 1 for the planar model = –1 for the nonplanar model
EID	Integer > 0	External aerodynamic box identification number
INTEID	Integer > 0	Internal aerodynamic box identification number (aerodynamic degree of freedom number) see Remark 2
MACROID	Integer > 0	Macroelement identification number on which the box lies
MACROTYP	Text (8)	Macroelement type (e.g. PAERO6, CAERO6)
CMPNT	Text (8)	Component type (FIN, CANARD, WING, FUSEL, of POD)
NGRID	Integer > 0	Number of grids connected to the box = 4 or 3 for panels = 2 for airfoil line segments
GRID1	Integer > 0	Grid identification number of an AEROGEOM grid for inboard/upstream location
GRID2	Integer > 0	Grid identification number of an AEROGEOM grid for inboard/downstream location
GRID3	Integer > 0	Grid identification number of AEROGEOM grid for outboard/downstream location
GRID4	Integer ≥ 0	Grid identification number of AEROGEOM grid for outboard/upstream location

Created By: STEADY and/or STEADYNP modules

Notes:

1. The grid points referred to by this relation are stored in the AEROGEOM entity.
2. Airfoil geometry is also defined by this relation but the "elements" are line segments not related to the control points of the panel model. For these elements, the internal identification number is set to – 1 rather than the degree of freedom identifier in solution matrices.
3. Airfoil geometry is defined by any and all elements with NGRID=2.

Entity: **CAROUGEO**

Entity Type: Relation

Description: Contains the connectivity data for the aerodynamic planform of the flutter analysis model. These elements are not used for data recovery, but can be used in combination with the "grids" in AERUGEOM to create an ASTROS FE model using ROD's and QUAD's that represents the paneling of the aero model.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EXTID	Integer > 0	External aerodynamic box identification number
INTEID	Integer > 0	Internal aerodynamic box identification number (aerodynamic degree of freedom number) see Remark 2
MACROID	Integer > 0	Macroelement identification number on which the box lies
NGRID	Integer > 0	Number of grids connected to the box = 4 or 3 for panels = 2 for airfoil line segments
MACROTYP	Text (8)	Macroelement type (e.g. PAERO6, CAERO6)
IGID	Integer > 0	Interference group identification
GRID1	Integer > 0	Grid identification number of an AEROGEOM grid for inboard/upstream location
GRID2	Integer > 0	Grid identification number of an AEROGEOM grid for inboard/downstream location
GRID3	Integer > 0	Grid identification number of AEROGEOM grid for outboard/downstream location
GRID4	Integer ≥ 0	Grid identification number of AEROGEOM grid for outboard/upstream location

Created By: UNSTEADY module

Notes:

1. The grid points referred to by this relation are stored in the AERUGEOM entity.
2. Airfoil geometry is also defined by this relation but the "elements" are line segments not related to the control points of the panel model. For these elements, the internal identification number is set to – 1 rather than the degree of freedom identifier in solution matrices.
3. Airfoil geometry is defined by any and all elements with NGRID=2.

Entity: **CASE**

Entity Type: Relation

Description: Contains the case parameters for each analysis within each boundary condition as input in the solution control packet.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
OAFLAG	Integer > 0	Optimize/analyze flag = 1 Optimize = 2 Analyze
BCID	Integer > 0	Subcase identification number
CASEID	Integer > 0, Key	Boundary condition identification number
MPCSETID	Integer	Multipoint constraint set identification number
SPCSETID	Integer	Single point constraint set identification number
REDSETID	Integer	Guyan reduction constraint set identification number
SUPSETID	Integer	Support set identification number
METHOD	Integer	Real eigenvalue extraction method set identification number
CMETHOD	Integer	Complex eigenvalue extraction method set identification number
DYNRED	Integer	Dynamic reduction set identification number
INERTIA	Integer	Inertia relief mode shapes set identification number
TFSETID	Integer	Transfer function set identification number
K2PP	Text (8)	K2PP name
M2PP	Text (8)	M2PP name
B2PP	Text (8)	B2PP name
K2GG	Text (8)	K2GG name
M2GG	Text (8)	M2GG name

NAME	TYPE/KEY	DESCRIPTION
DISFLAG	Integer	Discipline flag = 1 Statics = 2 Modes = 3 Saero = 4 Flutter = 5 Transient = 6 Frequency = 7 Buckling = 8 Blast = 9 Nonplanar Saero
MECHLOAD	Integer	Mechanical load set identification number
THRMLOAD	Integer	Thermal load set identification number
GRAVLOAD	Integer	Gravity load set identification number
TRIMID	Integer	Trim set identification number
TRIMSYM	Integer	Trim symmetry flag = –1 Antisymmetric = 0 Asymmetric = 1 Symmetric
DCONST	Integer	Design constraint set identification number
DCSTRESS	Integer	Stress constraint set identification number
DCSTRAIN	Integer	Strain constraint set identification number
DCONFID	Integer	DCONF identifiction number
DLOADID	Integer	Dynamic load set identification number
DRMETH	Integer	Dynamic response method = 1 Direct = 2 Modal
TIMESTEP	Integer	Time step set identification number
FREQSTEP	Integer	Frequency step set identification number
FFTID	Integer	Fast Fourier transform set identification number
GUSTID	Integer	Gust set identification number
INITCON	Integer	Initial condition set identification number
RANDOMID	Integer	Random set identification number
BLASTID	Integer	Blast set identification number
BUCKLEID	Integer	Buckling eigenvalue extraction set identification number
FLUTID	Integer	Flutter set identification number

NAME	TYPE/KEY	DESCRIPTION
CONTROL	Text (8)	Name of aerodynamic extra point splining matrix
DAMPID	Integer	Damping set identification number
ESET	Integer	Extra point set identification number
ACCEPRNT	Integer vector (20)	Acceleration print selection WORD 1 Print set identification number > 0, or = 0 NONE = –1 ALL = –2 LAST WORD 2 Punch set identification number WORD 3 Print form = 0 Rectangular = 1 Polar WORD 4 Punch form WORD 5 Print frequency set identification number WORD 6 Punch frequency set identification number WORD 7 Print iteration set identification number WORD 8 Punch iteration set identification number WORD 9 Print mode set identification number WORD 10 Punch maode set identification number WORD 11 Print time set identification number WORD 12 Punch time set identification number
AIRDPRNT	Integer vector (20)	Aerodynamic displacement print selection
DISPPRNT	Integer vector (20)	Displacement print selection WORD 1 Print set identification number WORD 2 Punch set identification number WORD 3 Print form = 0 Rectangular = 1 Polar WORD 4 Punch form
ENERPRNT	Integer vector (20)	Strain energy print selection
FORCPRNT	Integer vector (20)	Element force print selection
GPFOPRNT	Integer vector (20)	Grid point force print selection

NAME	TYPE/KEY	DESCRIPTION
GPWGPRNT	Integer vector (20)	Grid point weight generation print selection
LOADPRNT	Integer vector (20)	Load print selection
MASSPRNT	Integer vector (20)	Mass matrix print selection
MPCFPRNT	Integer vector (20)	Multi-point constraint force print selection
QHHPRNT	Integer vector (20)	QHH matrix print selection
QHJPRNT	Integer vector (20)	QHJ matrix print selection
ROOTPRNT	Integer vector (20)	Flutter and normal modes eigenvalue print selection
SPCFPRNT	Integer vector (20)	Single point constraint force print selection
STIFPRNT	Integer vector (20)	Stiffness matrix print selection
STRAPRNT	Integer vector (20)	Strain print selection WORD 13 Print layer/laminate option WORD 14 Punch layer/laminate option = 0 Layer strain = 1 Laminate strain = 2 Both layer and laminate strain
STREPRNT	Integer vector (20)	Stress print selection WORD 13 Print layer/laminate option WORD 14 Punch layer/laminate option = 0 Layer stress = 1 Laminate stress = 2 Both layer and laminate stress
TPREPRNT	Integer vector (20)	Trim pressure coefficient print selection
VELOPRNT	Integer vector (20)	Velocity print selection
TRIMPRNT	Integer	Steady aeroelastic trim print toggle
TITLE	Text (72)	User label TITLE
SUBTITLE	Text (72)	User label SUBTITLE
LABEL	Text (72)	User label LABEL

Created By: Module SOLUTION

Notes:

1. The format of the ACCEPRNT vector is typical of the format of all the print selection vectors. Additionally, the format for the print set Identification number in the ACCEPRNT vector is typical of that of the other set Identification numbers in the vector.
2. The CASE, JOB and OPTIMIZE relation entities together contain the solution control requests as input in the solution control packet. CASE contains the case-dependent parameters, JOB contains the case-independent requests and OPTIMIZE contains the optimization-dependent requests.

Entity: **CASELIST**

Entity Type: Relation

Description: Contains the list of subcase identification numbers as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer	Set identification number
CASEID	Integer	Subcase identification number

Created By: Module IFP.

Entity: **CBAR**

Entity Type: Relation

Description: Contains the element connectivity data for the BAR element as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
PID1	Integer > 0	Property identification number of a PBAR tuple
GRID1	Integer > 0	Grid point identification for end A
GRID2	Integer > 0	Grid point identification for end B
GRID3	Integer ≥ 0	Grid point identification for orientation vector definition
ORIENTX	Real	Orientation vector
ORIENTY	Real	
ORIENTZ	Real	
TMAX	Real	Maximum area for design
PINA	Integer ≥ 0	Components pinned at end A
PINB	Integer ≥ 0	Components pinned at end B
OFFSETAX	Real	Offsets from GRID1 and GRID2 to the ends of the beam element
OFFSETAY	Real	
OFFSETAZ	Real	
OFFSETBX	Real	
OFFSETBY	Real	
OFFSETBZ	Real	

Created By: Module IFP

Notes:

1. This relation is used by the MAKEST module to build the BEAMEST relation.

Entity: **CELAS1**

Entity Type: Relation

Description: Contains the element connectivity data for the scalar spring element as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0	Element identification number
PID1	Integer > 0	Identification number of a PELAS property entry
GRID1	Integer ≥ 0	Grid or scalar point identification number
COMPNTS1	6 ≥ Integer ≥ 0	Component number
GRID2	Integer ≥ 0	Grid or scalar point identification number
COMPNTS2	6 ≥ Integer ≥ 0	Component number
TMAX	Real	Maximum spring constant value for design

Created By: Module IFP

Notes:

1. This relation is used by the MAKEST module to build the ELASEST relation.

Entity: **CELAS2**

Entity Type: Relation

Description: Contains the element connectivity and property data for the scalar spring element as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0	Element identification number
K	Real	The value of the scalar spring constant
GRID1	Integer ≥ 0	Grid or scalar point identification number
COMPNTS1	6 ≥ Integer ≥ 0	Component number
GRID2	Integer ≥ 0	Grid or scalar point identification number
COMPNTS2	6 ≥ Integer ≥ 0	Component number
DAMPCOEF	Real	Damping coefficient
STRSCOEF	Real	Stress coefficient
TMIN	Real	Minimum spring constant value for design
TMAX	Real	Maximum spring constant value for design

Created By: Module IFP

Notes:

1. This relation is used by the MAKEST module to build the ELASEST relation.

Entity: **CENTRESP**

Entity Type: Relation

Description: Contains the user function requested element centroidal coordinate values.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
VALUE	Real	Response value

Created By: Module PFBULK

Entity: **CIHEX1**

Entity Type: Relation

Description: Contains the element connectivity data for the linear isoparametric hexahedron element as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
PID	Integer > 0	Identification number of property card
GRID1, GRID2	Integer > 0	Grid point identification numbers defining the element geometry
GRID3, GRID4	Integer > 0	
GRID5, GRID6	Integer > 0	
GRID7, GRID8	Integer > 0	

Created By: Module IFP

Notes:

1. This relation is used by the MAKEST module to build the IHEX1EST relation.

Entity: **CIHEX2**

Entity Type: Relation

Description: Contains the element connectivity data for the quadratic isoparametric hexahedron element as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
PID	Integer > 0	Identification number of property card
GRID1, GRID2	Integer > 0	Grid point identification numbers defining the element geometry
GRID3, GRID4	Integer > 0	
GRID5, GRID6	Integer > 0	
GRID7, GRID8	Integer > 0	
GRID9, GRID10	Integer > 0	
GRID11, GRID12	Integer > 0	
GRID13, GRID14	Integer > 0	
GRID15, GRID16	Integer > 0	
GRID17, GRID18	Integer > 0	
GRID19, GRID20	Integer > 0	

Created By: Module IFP

Notes:

1. This relation is used by the MAKEST module to build the IHEX2EST relation.

Entity: **CIHEX3**

Entity Type: Relation

Description: Contains the element connectivity data for the cubic isoparametric hexahedron element as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
PID	Integer > 0	Identification number of property card
GRID1, GRID2	Integer > 0	Grid point identification numbers defining the element geometry
GRID3, GRID4	Integer > 0	
GRID5, GRID6	Integer > 0	
GRID7, GRID8	Integer > 0	
GRID9, GRID10	Integer > 0	
GRID11, GRID12	Integer > 0	
GRID13, GRID14	Integer > 0	
GRID15, GRID16	Integer > 0	
GRID17, GRID18	Integer > 0	Grid point identification numbers defining the element geometry
GRID19, GRID20	Integer > 0	
GRID21, GRID22	Integer > 0	
GRID23, GRID24	Integer > 0	
GRID25, GRID26	Integer > 0	
GRID27, GRID28	Integer > 0	
GRID29, GRID30	Integer > 0	
GRID31, GRID32	Integer > 0	

Created By: Module IFP

Notes:

1. This relation is used by the MAKEST module to build the IHEX3EST relation.

Entity: **CLAMBDA**

Entity Type: Relation

Description: Contains results of a flutter analysis for a series of boundary conditions, Mach numbers and atmospheric densities.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
NITER	Integer	Iteration number
BCID	Integer	The boundary condition number
MACH	Real	Mach number of the flutter analysis
RHOREF	Real	Reference atmospheric density
RHO	Real	Atmospheric density of the flutter analysis
VELOCITY	Real	True velocity of the flutter analysis
FSID	Integer	Flutter set identification
SCNUM	Integer	Flutter subcase identification number
MODENO	Integer	Mode number associated with the flutter
RLAMB	Real	Real part of the flutter eigenvalue
ILAMB	Real	Imaginary part of the flutter eigenvalue
DAMPVAL	Real	Damping ratio
OMEGA	Real	Frequency in radians per second of the flutter eigenvalue = 2 * VELOCITY * ILAMB / REFB
PNUM	Integer	Pointer to CONST tuple for the associated constraint

Created By: Module FLUTTRAN

Notes:

1. The reference semichord for the unsteady area model is stored as the eleventh word of the INFO array.

Entity: **CMASS1**

Entity Type: Relation

Description: Contains the element connectivity data for the scalar mass element as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
PID1	Integer > 0	Property identification number of a PMASS tuple
GRID1	Integer > 0	Grid or scalar point identification number
COMPNTS1	Integer ≥ 0	Component of GRID1 to which the element is connected
GRID2	Integer ≥ 0	Grid or scalar point identification number
COMPNTS2	Integer ≥ 0	Component of GRID2 to which the element is connected
TMAX	Real	Maximum mass for design

Created By: Module IFP

Notes:

1. This relation is used by the MAKEST module to build the MASSEST relation.

Entity: **CMASS2**

Entity Type: Relation

Description: Contains the element connectivity data for the scalar mass element as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
MASS	Real	The value of the scalar mass
GRID1	Integer > 0	Grid or scalar point identification number
COMPNTS1	Integer ≥ 0	Component of GRID1 to which the element is connected
GRID2	Integer ≥ 0	Grid or scalar point identification number
COMPNTS2	Integer ≥ 0	Component of GRID2 to which the element is connected
TMIN	Real	Minimum mass for design
TMAX	Real	Maximum mass for design

Created By: Module IFP

Notes:

1. This relation is used by the MAKEST module to build the MASSEST relation.

Entity: **CONEFFF**

Entity Type: Relation

Description: Contains the definition of adjustment factors for control surface effectiveness values for use in flutter analysis.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Effectiveness identification number
EFFVAL	Real	Effectiveness value
MODE	Integer > 0	Structural mode to which the effectiveness is to be applied
MACROID	Integer	aerodynamic component (macroelement) on which the control surface lies
BOX1, BOX2	Integer > 0	First and last box whose effectiveness is to be altered

Created By: Module IFP

Entity: **CONEFFS**

Entity Type: Relation

Description: Contains the definition of adjustment factors for control surface effectiveness values for use in static aeroelastic analysis and nonplanar aerodynamic analysis.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Effectiveness identification number
LABELI	Text (8)	Control surface label
EFFI	Real	Effectiveness value for the associated surface

Created By: Module IFP

Entity: **CONLINK**

Entity Type: Relation

Description: Contains the control surfaces and participation factors specified on the CONLINK Bulk Data entry.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
LABEL	Text (8)	Label of the control surface that is made up of a combination of other conatro surfaces
LABELI	Text (8)	Label of control surface defined by AESURF
VALUEI	Real	Participation factor

Created By: Module IFP

Entity: **CONM1**

Entity Type: Relation

Description: Contains the element data for a 6 x 6 symmetric mass matrix at a grid point as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
GRID1	Integer > 0	Grid point identification number
CID1	Integer ≥ 0	Coordinate system identification number for matrix coordinate system
M11, M21, M22	Real	Elements of the 6x6 symmetric mass matrix
M31, M32, M33	Real	
M41, M42, M43	Real	
M44	Real	
M51, M52, M53	Real	
M54, M55	Real	
M61, M62, M63	Real	
M64, M65, M66	Real	

Created By: Module IFP

Notes:

1. This relation is used by the MAKEST module to build the CONM1EST relation.

Entity: **CONM1EST**

Entity Type: Relation

Description: Contains the element summary data for a concentrated mass defined in the CONM1 relation.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
SIL1	Integer > 0	Internal grid point identification number
CIDMASS	Integer ≥ 0	Coordinate system identification number for matrix coordinate system
M11, M21, M22	Real	Elements of the 6x6 symmetric mass matrix
M31, M32, M33	Real	
M41, M42, M43	Real	
M44	Real	
M51, M52, M53	Real	
M54, M55	Real	
M61, M62, M63	Real	
M64, M65, M66	Real	
CORD1	Integer ≥ 0	Coordinate system of SIL1
X, Y, Z	Real	Basic coordinates of SIL1

Created By: Module MAKEST

Notes:

1. This relation is built from the CONM1 and grid relations. It contains one tuple for each concentrated mass element defined in the CONM1 relation.

Entity: **CONM2**

Entity Type: Relation

Description: Contains the element data for a concentrated mass at a structural grid point as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
GRID1	Integer > 0	Grid point identification number
CID1	Integer	Coordinate system identification number
MASS	Real	Value of the concentrated mass
X1, X2, X3	Real	Components of offset from GRID1 to the mass
I11, I21, I22	Real	Mass moments of inertia
I31, I32, I33	Real	
TMIN	Real	Minimum mass for design
TMAX	Real	Maximum mass for design

Created By: Module IFP

Notes:

1. This relation is used by the MAKEST module to build the CONM2EST relation.

Entity: **CONM2EST**

Entity Type: Relation

Description: Contains the element summary data for a concentrated mass element defined in the CONM2 relation.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
SIL1	Integer > 0	Internal grid point identification number
CIDMASS	Integer ≥ 0	Coordinate system identification number
MASS	Real	Mass value
OFFSETX	Real	Offsets from SIL1 to mass (see note 1)
OFFSETY	Real	
OFFSETZ	Real	
I11, I21,I22	Real	Mass moments of inertia (see Note 1)
I31, I32, I33	Real	
COORD1	Integer ≥ 0	Displacement coordinate system for SIL1
X, Y, Z	Real	Basic coordinates of SIL1
DESIGN	Integer	Design flag
NLFLAG	Integer ≥ 0	Design variable nonlinear flag = 0 Linear = 1 $I_{ij} \neq 0$

Created By: Module MAKEST

Notes:

1. Refer to the CONM2 Bulk Data Entry for further details on the definition of the OFFSET and Iij terms.
2. This relation is built from the CONM2 grid relations. It contains one tuple for each concentrated mass element defined in the CONM2 relation.

Entity: **CONROD**

Entity Type: Relation

Description: Contains the connectivity and property data for a ROD element as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
GRID1	Integer > 0	Grid point identification number for end A
GRID2	Integer > 0	Grid point identification number for end B
MID1	Integer > 0	Material property identification number
AREA	Real ≥ 0	Element cross-sectional area
TORSION	Real ≥ 0	Element torsional constant
STRSCOEF	Real	Stress recovery factor
NSM	Real ≥ 0	Element nonstructural mass
TMIN	Real ≥ 0	Minimum cross-sectional area in design
TMAX	Real ≥ 0	Maximum cross-sectional area in design

Created By: Module IFP

Notes:

1. This relation is used by the MAKEST module to build the RODEST relation.

Entity: **CONST**

Entity Type: Relation

Description: Contains the constraint values and constraint sensitivity processing data.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
NITER	Integer > 0	Iteration number
CVAL	Real	Constraint value
CTYPE	Integer > 0	Constraint type (see Note 2 below)
BCID	Integer > 0 or NULL	Boundary condition identification number for constraint value if boundary condition dependent Non-boundary dependent constraints are: minimum thickness (CTYPE=1) maximum thickness (CTYPE=2) laminate composition (CTYPE=13) laminate min. gauge (CTYPE=14) ply min. gauge (CTYPE=15) BAR dimension relation (CTYPE=18)
DISFLAG	Integer > 0 or NULL	Discipline type flag from CASE relation (where appropriate) Non-discipline dependent constraints are: minimum thickness (CTYPE=1) maximum thickness (CTYPE=2) laminate composition (CTYPE=13) laminate min. gauge (CTYPE=14) ply min. gauge (CTYPE=15) BAR dimension relation (CTYPE=18)
ACTVFLAG	Integer > 0 or NULL	Flag denoting status of the constraint as active (=1) or inactive (=0) ACTVFLAG will have NULL value prior to constraint screening in ACTCON The entry with CLASS as OBJECTIVE has its ACTVFLAG set to be active
SCNUM	Integer > 0 or NULL	See Remark 11
PNUM	Integer > 0 or NULL	See Remark 12
SUBSCRPT	Integer > 0 or NULL	Subscript number for SAERO discipline constraints of types 3, 4, 5, 6, 9, 10, 11, and 12
DISPCOL	Integer > 0 or NULL	Column number in the matrix of pseudodisplacements/accelerations for static aeroelastic constraints ot types 9, 10, and 12
ETYPE	Text (8) or NULL	Element type used for stress/strain and thickness constraints

NAME	TYPE/KEY	DESCRIPTION
EID	Integer or NULL	Element identification number
LAYERNUM	Integer or NULL	Element layer information (See Remark 14)
DVSYMBL	Text(8)	Designed dimension symbol (D1, D2, D3, D4, D5, D6, D7, D8, D9, D10 for BAR element cross section dimension; A for element area, T for element thickness)
SCON	Integer > 0 or NULL	See Remark 13
VSCON	Real vector(6)	Allowables for stress/strain constraints
BENDPOST	Integer	Bending position flag for nonlinear sensitivity
SENSPRM1	Real	General values useful for sensitivity calculations (see Remark 10)
SENSPRM2	Real	
SENSPRM3	Real	
SENSPRM4	Text (8)	
SENSPRM5	Real vector(10)	
PRINTKEY	Integer	Pointer to the GRADIENT entity containing the gradient of the constraint with respect to the global variables = 0 if no gradient was stored (see Remark 15)
INSTANCE	Integer	User function instance number
CLASS	Text(16)	User function class (OBJECTIVE/CONSTRAINTS)
UFNAME	Text(8)	User function name

Created By: See Note below.

Notes:

1. NULL values imply the value is supplied with the database default null value. These are typically bit patterns which represent illegal values of their respective data types. See RENULi utility documentation. For this relation, zero is sometimes used in place of the database NULL value.
2. The constraint types are:
   0 = Objective function
   1 = Minimum thickness constraint
   2 = Maximum thickness constraint
   3 = Displacement constraint
   4 = Stress constraint
   5 = Strain constraint on Ex principal strain
   6 = Strain constraint on Ey principal strain
   7 = Frequency constraint
   8 = Flutter constraint
   9 = Lift Effectiveness Constraint

10 = Aileron Effectiveness Constraint
11 = Trim Parameter Limit Value Constraint (DCONTRM)
12 = Stability Derivative Constraint (DCONSCF)
13 = Laminate Composition Constraint (DCONLAM)
14 = Laminate Minimum Gauge Constraint (DCONLMN)
15 = Ply Minimum Gauge Constraint (DCONPMN)
16 = Panel Buckling Constraint (DCONBK)
17 = Euler Buckling Constraint (DCONBKE)
18 = BAR Element Dimemsion Relation Constraint(DCONSDE)
19 = User Function Constraint (DCONF)

3. Constraints of Types 1 and 2 are evaluated in the TCEVAL module. The sensitivities are evaluated in the MAKDFV module.
4. Constraints of Type 3 are evaluated in the DCEVAL module. The sensitivities are evaluated in the MAKDFU module.
5. Constraints of Type 4, 5 and 6 are evaluated in the SCEVAL module. The sensitivities are evaluated in the MAKDFU module.
6. Constraints of Type 7 are evaluated in the FCEVAL module. The sensitivities are evaluated in the FREQSENS module.
7. Constraints of Type 8 are evaluated in the FLUTTRAN module. The sensitivities are evaluated in the FLUTSENS module.
8. Constraints of Types 9, 10, 11, and 12 are evaluated in the SAERO module. The sensitivities of 9, 10, and 12 are evaluated in the AEROEFFS and those of 11 in the AEROSENS module.
9. Constraints of Types 13, 14, and 15 are evaluated in the LAMINCON module. The sensitivities are evaluated in the LAMINSNS module.
   Constraints of Types 16 and 17 are evaluated in the modules PBKLEVAL and EBKLEVAL, respectively. The sensitivities are evaluated in the modules PBKLSENS and EBKLSENS, respectively.
   Constraints of Types 18 are evaluated in the BCEVAL module. The sensitivities are evaluated in the MKDFDV module.
   User function constraints (Types 18) are evaluated in the FNEVAL module. The sensitivities are evaluated in the DESIGN module.
10. The SENSPRM1, 2, 3, 4 and 5 attributes contain values useful in sensitivity analysis for certain constraint types.
    Type 1 - SENSPRM1 contains the minimum gauge used to normalize the constraint.
    Type 2 - SENSPRM2 contains the maximum gauge used to normalize the constraint.
    Type 7 - SENSPRM1 contains the current value of the associated eigenvalue.
    Type 9 - SENSPRM1 contains the current value of the associated rigid lift curve slope. SENSPRM2 contains the value of the associated required ratio.
    Type 10 - SENSPRM1 contains the current value of the associated dimensional flexible rolling moment slope due to aileron deflection.
    SENSPRM2 contains the current value of the associated flexible rolling moment slope due to roll rate.
    SENSPRM3 contains the required roll effectiveness and other constants:

$$\text{SENSPRM3} = \frac{b * 180}{2\varepsilon_{RQ}}$$

where
b = reference span
$2\varepsilon_{RQ}$ = required aileron effectiveness

SENSPRM4 contains the name of the rolling control surface whose effectiveness is constrained.

Type 11 - SENSPRM1 contains the required value of the trim parameter.
SENSPRM4 contains the name of the trim parameter whose derivative is constrained.

Type 12 - SENSPRM1 contains the required dimensional value of the stability derivative.
SENSPRM2 contains the real equivalent of the degree of freedom number (1, 2, ... or 6) representing the DOF associated with the derivative.
SENSPRM4 contains the name of the trim parameter or acceleration whose derivative is constrained.

Type 13 - SENSPRM1 contains the required upper or lower bound percentage of ply to laminate thickness.
SENSPRM2 contains the current ply thickness
SENSPRM3 contains the current laminate thickness

Type 14 - SENSPRM1 contains the minimum thickness value.
SENSPRM3 contains the current laminate thickness

Type 15 - SENSPRM1 contains the minimum thickness value.
SENSPRM2 contains the current ply thickness

Type 17 - SENSPRM4 contains the PBAR1 indicator.
SENSPRM5 contains the buckling sensitivity information related with each BAR element cross section dimension.

Type 18 - SENSPRM5 contains the required cross section dimension factors.

11. The SCNUM attribute contains general information to allow computation of the sensitivities. These data are the following:

Type 1 - = 0 if constraint does not appear on a DCONTHK/2 entry
= 1 if it does appear on DCONTHK/2

Type 2 - NULL

Type 3 - Subcase number of discipline generating the constraint

Type 4 - Subcase number of discipline generating the constraint

Type 5 - Subcase number of discipline generating the constraint

Type 6 - Subcase number of discipline generating the constraint

Type 7 - Mode number associated with the constraint

Type 8 - Subcase number generating the constraint

Type 9 - Subcase number generating the constraint

Type 10 - Subcase number generating the constraint

Type 11 - Subcase number generating the constraint

Type 12 - Subcase number generating the constraint

Type 13 - NULL

Type 14 - NULL

Type 15 - NULL

Type 16 - Subcase number generating the constraint
Type 17 - Subcase number generating the constraint
Type 18 - NULL
Type 19 - NULL

12. The PNUM attribute contains general pointer information to allow computation of the sensitivities. The pointer data are the following:
Type 1 - PMINT matrix column number associated with the constraint
Type 2 - PMAXT matrix column number associated with the constraint
Type 3 - Displacement constraint number which points into the DCENT entity
Type 4 - Row in GLBSIG where first stress component for the element is stored
Type 5 - Row in GLBSIG where first stress component for the element is stored
Type 6 - Row in GLBSIG where first stress component for the element is stored
Type 7 - NULL
Type 8 - Count number in a running count of flutter roots. Matches the PNUM attribute in CLAMBDA
Type 9 - NULL
Type 10 - NULL
Type 11 - NULL
Type 12 - NULL
Type 13 - Defines the *laminate*
= 0 if the lamanate thickness comprises all layers
= LAMSET id of PLYLIST data if the lamanate thickness comprises a subset of layers
Type 14 - = 0
Type 15 - = 0
Type 16,17 - Count number matches the PNUM attribute in PDLIST
Type 18 - = 0
Type 19 - = 0

13. The SCON attribute contains general information to allow computation of the sensitivities. These data are the following:
Type 1 - NULL
Type 2 - NULL
Type 3 - DCONDSP/DCID value
Type 4 - = 1 for Von Mises Stress
= 2 for Tsai Wu Stress
Type 5 - = +3 Principal strain constraint using tension allowable
= – 3 Principal strain constraint using compression allowable
= +4 Fiber/transverse strain constraint using tension allowable
= – 4 Fiber/transverse strain constraint using compression allowable
Type 6 - = +3 Principal strain constraint using tension allowable
= – 3 Principal strain constraint using compression allowable
= +4 Fiber/transverse strain constraint using tension allowable
= – 4 Fiber/transverse strain constraint using compression allowable
Type 7 - UPPER (=1) or LOWER (=–1) bound flag

Type 8 - A combined number noting the velocity, mode and subcase number generating the constraint of the form:

xxxyyyzzz

where

xxx = subcase number
yyy = mode number
zzz = velocity number

each are limited to 999. This value is only useful in that sorting by SCON sorts the constraints by velocity within each mode within each subcase.

Type 9 - UPPER (=1) or LOWER (=–1) bound flag
Type 10 - UPPER (=1) or LOWER (=–1) bound flag
Type 11 - UPPER (=1) or LOWER (=–1) bound flag
Type 12 - UPPER (=1) or LOWER (=–1) bound flag
Type 13 - UPPER (=1) or LOWER (=–1) bound flag
Type 14 - =0
Type 15 - =0
Type 16 - =0
Type 17 - =0
Type 18 - =0
Type 19 - =0

14. The LAYERNUM is NULL except for:
Types 1, 4, 5, and 6 –contain the layer number (if applicable) or 0
Types 13, and 15 – if > 0, contain the layer number of the *ply*
if < 0, contain the PLYLIST id of the set of layers in the *ply*
15. The PRINTKEY is only set if the gradient is stored. This is done only when the requested objective and/or constraint fradient is selected in a print of punch request.

Entity: **CONSTORD**

Entity Type: Relation

Description: Contains the reorded constraint values and constraint sensitivity processing data for the current design iteration.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
NITER	Integer > 0	Iteration number
CVAL	Real	Constraint value
CTYPE	Integer > 0	Constraint type (see Note 2 for CONST)
BCID	Integer > 0 or NULL	Boundary condition identification number for constraint value if boundary condition dependent Non-boundary dependent constraints are: minimum thickness (CTYPE=1) maximum thickness (CTYPE=2)
DISFLAG	Integer > 0 or NULL	Discipline type flag from CASE relation (where appropriate) Non-discipline dependent constraints are: minimum thickness (CTYPE=1) maximum thickness (CTYPE=2) laminate composition (CTYPE=13) laminate min. gauge (CTYPE=14) ply min. gauge (CTYPE=15)
SCNUM	Integer > 0 or NULL	See Remark 11 for CONST
PNUM	Integer > 0 or NULL	See Remark 12 for CONST
SUBSCRPT	Integer > 0 or NULL	Subscript number for SAERO discipline constraints of types 3, 4, 5, 6, 9, 10, 11, and 12
SCON	Integer > 0 or NULL	See Remark 13 for CONST
SENSPRM4	Text (8)	String for sensitivity
PRINTKEY	Integer	Pointer to the GRADIENT entity containing the gradient of the constraint with respect to the global variables = 0 if no gradient was stored (see Remark 15 for CONST)

Created By: See Note in preceding CONST description.

Entity: **CONVERT**

Entity Type: Relation

Description: Contains the conversion factors for various physical quantities as input from the Bulk Data file.

Relation Attributes:

NAME	Type/KEY	DESCRIPTION
QUANTITY	Test (8)	Character string identifying the physical quantity whose units are to be converted
FACTOR	Real	Conversion factor to be applied

Created By: Module IFP

Notes:

1. Refer to CONVERT Bulk Data entry for the valid QUANTITY values.

Entity: **CORDRESP**

Entity Type: Relation

Description: Contains the user function requested grid point coordinate values.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
VALUE	Real	Response value

Created By: Module PFBULK

Entity: **CORD1C**

Entity Type: Relation

Description: Contains the coordinate system definition for a cylindrical coordinate system as input from the Bulk Data file.

Relation Attributes:

NAME	Type/KEY	DESCRIPTION
CID1	Integer > 0	Coordinate system identification number
GRID1	Integer > 0	The grid point identification number which locates the system origin
GRID2	Integer > 0	The grid point identification number which defines the system z-axis
GRID3	Integer > 0	The grid point identification number which defines a point lying in the system xz-plane

Created By: Module IFP

Notes:

1. This relation is used by the MKTMAT module to build the CSTM relation.

Entity: **CORD2C**

Entity Type: Relation

Description: Contains the coordinate system definition for a cylindrical coordinate system as input from the Bulk Data file.

Relation Attributes:

NAME	Type/KEY	DESCRIPTION
CID1	Integer > 0	Coordinate system identification number
RID	Integer ≥=0	Coordinate system identification number of system in which the coordinates of the defining locations are given
A1, A2, A3	Real	Coordinates of system origin
B1, B2, B3	Real	Coordinates defining z-axis
C1, C2, C3	Real	Coordinates defining xz plane

Created By: Module IFP

Notes:

1. This relation is used by the MKTMAT module to build the CSTM relation.

Entity: **CORD1R**

Entity Type: Relation

Description: Contains the coordinate system definition for a rectangular coordinate system as input from the Bulk Data file.

Relation Attributes:

NAME	Type/KEY	DESCRIPTION
CID1	Integer > 0	Coordinate system identification number
GRID1	Integer > 0	The grid point identification number which locates the system origin
GRID2	Integer > 0	The grid point identification number which defines the system z-axis
GRID3	Integer > 0	The grid point identification number which defines a point lying in the system xz-plane

Created By: Module IFP

Notes:

1. This relation is used by the MKTMAT module to build the CSTM relation.

Entity: **CORD2R**

Entity Type: Relation

Description: Contains the coordinate system definition for a rectangular coordinate system as input from the Bulk Data file.

Relation Attributes:

NAME	Type/KEY	DESCRIPTION
CID1	Integer > 0	Coordinate system identification number
RID	Integer ≥ 0	Coordinate system identification number of system in which the coordinates of the defining locations are given
A1, A2, A3	Real	Coordinates of system origin
B1, B2, B3	Real	Coordinates defining z-axis
C1, C2, C3	Real	Coordinates defining xz-plane

Created By: Module IFP

Notes:

1. This relation is used by the MKTMAT module to build the CSTM relation.

Entity: **CORD1S**

Entity Type: Relation

Description: Contains the coordinate system definition for a spherical coordinate system as input from the Bulk Data file.

Relation Attributes:

NAME	Type/KEY	DESCRIPTION
CID1	Integer > 0	Coordinate system identification number
GRID1	Integer > 0	The grid point identification number which locates the system origin
GRID2	Integer > 0	The grid point identification number which defines the system z-axis
GRID3	Integer > 0	The grid point identification number which defines a point lying in the system xz-plane

Created By: Module IFP

Notes:

1. This relation is used by the MKTMAT module to build the CSTM relation.

Entity: **CORD2S**

Entity Type: Relation

Description: Contains the coordinate system definition for a spherical coordinate system as input from the Bulk Data file.

Relation Attributes:

NAME	Type/KEY	DESCRIPTION
CID1	Integer > 0	Coordinate system identification number
RID	Integer ≥ 0	Coordinate system identification number of system in which the coordinates of the defining locations are given
A1, A2, A3	Real	Coordinates of system origin
B1, B2, B3	Real	Coordinates defining z-axis
C1, C2, C3	Real	Coordinates defining xz-plane

Created By: Module IFP

Notes:

1. This relation is used by the MKTMAT module to build the CSTM relation.

Entity: **CQDMEM1**

Entity Type: Relation

Description: Contains the element connectivity data for the linear isoparametric quadrilateral membrane element as input from the Bulk Data file.

Relation Attributes:

NAME	Type/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
PID1	Integer > 0	Property identification number of P-Type tuple
GRID1, GRID2	Integer > 0	Grid point identification number
GRID3, GRID4	Integer > 0	
CID	Integer ≥ 0	Coordinate system used to define material axis
THETA	Real	Material orientation angle for anisotropic materials
TMAX	Real ≥ 0	Maximun element thickness in design

Created By: Module IFP

Notes:

1. The PID refers to a PQDMEM1 tuple.
2. This relation is used by the MAKEST module to build the QDMM1EST relation.
3. Note that the relation contains two attributes CID and THETA to account for the dual definition of the THETA field on the CQDMEM1 bulk data entry.

Entity: **CQUAD4**

Entity Type: Relation

Description: Contains the element connectivity data for the quadrilateral bending element as input from the Bulk Data file.

Relation Attributes:

NAME	Type/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
PID1	Integer > 0	Property identification number of P-Type tuple
GRID1, GRID2	Integer > 0	Grid point identification number
GRID3, GRID4	Integer > 0	
CID1	Integer ≥ 0	Coordinate system used to define material orientation
THETA	Real	Material orientation angle for anisotropic materials
OFFSETO	Real	Offset of element reference plane from plane of the grid point
TMAX	Real ≥ 0	Maximun laminate thickness in design
THICK1	Real ≥ 0	Element thickness at grid point GRID1
THICK2	Real ≥ 0	Element thickness at grid point GRID2
THICK3	Real ≥ 0	Element thickness at grid point GRID3
THICK4	Real ≥ 0	Element thickness at grid point GRID4

Created By: Module IFP

Notes:

1. The PID may refer to a PCOMPi or PSHELL tuple.
2. This relation is used by the MAKEST module to build the QUAD4EST relation.
3. Note that the relation contains two attributes CID and THETA to account for the dual definition of the THETA field on the CQUAD4 bulk data entry.

Entity: **CROD**

Entity Type: Relation

Description: Contains the element connectivity data for the ROD element as input from the Bulk Data file.

Relation Attributes:

NAME	Type/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
PID1	Integer > 0	Property identification number of a PROD tuple
GRID1	Integer > 0	Grid point identification number defining end A
GRID2	Integer > 0	Grid point identification number defining end B
TMAX	Real ≥ 0	Maximum cross-sectional area in design

Created By: Module IFP

Notes:

1. This relation is used by the MAKEST module to build the RODEST relation.

Entity: **CSHEAR**

Entity Type: Relation

Description: Contains the connectivity data for the shear panel as input from the Bulk Data file.

Relation Attributes:

NAME	Type/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
PID1	Integer > 0	Property identification number of a PSHEAR tuple
GRID1, GRID2	Integer > 0	Grid point identification numbers defining the element geometry
GRID3, GRID4	Integer > 0	
TMAX	Real ≥ 0	Maximum thickness in design

Created By: Module IFP

Notes:

1. This relation is used by the MAKEST module to build the SHEAREST relation.

Entity: **CSTM**

Entity Type: Relation

Description: Contains the coordinate transformation matrices for all external coordinate systems.

Relation Attributes:

NAME	Type/KEY	DESCRIPTION
CID	Integer > 0, key	Unique coordinate system identification number
CORDTYPE	Integer > 0	The type of coordinate system
X0, Y0 ,Z0	Real	Basic coordinates of the system origin
T11, T21, T31	Real	Elements of the 3 x 3 orthogonal transformation matrix in column order
T12, T22, T32	Real	
T13, T23, T33	Real	

Created By: Module MKTMAT

Notes:

1. This relation contains one tuple for each external coordinate system in the problem.
2. The CORDTYPE attribute contains a value of:
   1 = if the system is rectangular
   2 = if the system is cylindrical
   3 = if the system is spherical

Entity: **CTRIA3**

Entity Type: Relation

Description: Contains the connectivity data for the triangular shell element as input from the Bulk Data file.

Relation Attributes:

NAME	Type/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
PID1	Integer > 0	Property tuple identification number
GRID1, GRID2	Integer > 0	Grid point identification numbers defining the element geometry
GRID3	Integer > 0	
CID1	Integer	Coordinate system used to define the material orientation
THETA	Real	Material orientation angle for anisotropic materials
OFFSET0	Real	Offset of element reference plane from plane of the grid point
TMAX	Real ≥ 0	Maximum laminate thicknesses at each grid point
THICK1	Real ≥ 0	Element thicknesses at each grid point
THICK2	Real ≥ 0	
THICK3	Real ≥ 0	

Created By: Module IFP

Notes:

1. The PID may refer to a PCOMPi or PSHELL tuple.
2. This relation is used by the MAKEST module to build the TRIA3EST relation.
3. Note that the relation contains two attributes CID and THETA in order to account for the dual definition of the THETA field on the CTRIA3 Bulk Data entry.

Entity: **CTRMEM**

Entity Type: Relation

Description: Contains the connectivity data for the constant strain triangular membrane element as input from the Bulk Data file.

Relation Attributes:

NAME	Type/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
PID1	Integer > 0	Property identification number of a PTRMEM tuple
GRID1	Integer > 0	Grid identification numbers defining the geometry
GRID2	Integer > 0	
GRID3	Integer > 0	
CID	Integer ≥ 0	Coordinate system used to define the material axis
THETA	Real	Material orientation angle for anisotropic materials
TMAX	Real ≥ 0	Maximum thickness in design

Created By: Module IFP

Notes:

1. This relation is used by the MAKEST module to build the TRMEMEST relation.
2. Note that the relation has two attributes CID and THETA to account for the dual definition of the THETA field on the CTRMEM bulk data entry.

Entity: **D**

Entity Type: Subscripted Matrix

Description: Contains the rigid body transformation matrix relating the displacements of the solution set to those of the support set.

Matrix Form: A variable sized design invariant matrix having one column for each degree of freedom in the support set and one row for each degree of freedom in the solution set for the current boundary condition.

Created By: MAPOL

Notes:

1. This matrix is design invariant and is, therefore, computed only once for each unrestrained boundary condition.

Entity: **DCENT**

Entity Type: Unstructured

Description: Contains collected displacement constraint information.

Record:

1. ID's of the NDSET displacement constraint sets.
i. Contains data for the (i-1)th constraint set. The information on each of these record is:

WORD #	VARIABLE	DESCRIPTION
1	SETID	From DCONDSP
2	NDCID	Number of constraints in this set
j	DCID	Displacement constraint ID
j+1	CTYPE	Constraint type (see Remark 4)
j+2	ALLOWD	Allowable
j+3	NTERMS	Number of terms in the constraint
k	INTID	Internal ID of constraint component
k+1	AJ	Factor on component

Notes:

1. There are NTERMS nested blocks of k data for each block of j data.
2. There are NDCID nested blocks of j data for each record.
3. There are NSET+1 records in the entity.
4. The constraint type is either UPPER bound (CTYPE=1) or LOWER bound (CTYPE=-1).

Entity: **DCONALE**

Entity Type: Relation

Description: Contains the roll effectiveness constraint definition as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Aerodynamic set identification of the imposed constraint
LABEL	Text (8)	Control surface label
CTYPE	Text (8)	Constraint type, either UPPER or LOWER
AEREQ	Real	The required roll effectiveness

Created By: Module IFP

Entity: **DCONBK**

Entity Type: Relation

Description: Contains the local panel buckling constraint definition as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer	Plate panel buckling constraint set identification
ETYPE	Text(8)	Element type (QUAD4, TRIA3)
EID	Integer > 0	Element identification number
LENGTH	Real > 0.0	Plate buckling panel length
WIDTH	Real > 0.0	Plate buckling panel width
CTYPE	Text (8)	Constraint type, either UPPER or LOWER
LAMREQ	Real	Buckling eigenvalue limit

Created By: Module IFP

Entity: **DCONBKE**

Entity Type: Relation

Description: Contains the Euler buckling constraint definition as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer	Plate panel buckling constraint set identification
ETYPE	Text(8)	Element type (BAR, ROD)
EID	Integer > 0	Element identification number
LENGTH	Real > 0.0	Rod buckling length
BC	Text(8)	Boundary conditions for control element
CTYPE	Text (8)	Constraint type, either UPPER or LOWER
LAMREQ	Real	Buckling eigenvalue limit
RSQR	Real	Inertia linking parameters for ROD
ALPHA		

Created By: Module IFP

Module IFPEntity: **DCONCLA**

Entity Type: Relation

Description: Contains the flexible lift curve slope constraint definition as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer	Aerodynamic set identification of the imposed constraint
CTYPE	Text (8)	Constraint type, either UPPER or LOWER
CLAREQ	Real	The required flexible lift curve slope ratio

Created By: Module IFP

Entity: **DCONDSP**

Entity Type: Relation

Description: Contains the design displacement constraint as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Constraint set identification number
DCID	Integer > 0	Constraint identification number
CTYPE	Text (8)	Constraint type, either UPPER or LOWER
ALLOWD	Real	Allowable displacement
LABEL	Text (8)	User defined label
GRIDI	Integer > 0	Grid point id to which constraint is applied
COMPNTI	Integer 1,2,3,4,5 or 6	Component of GRIDI
AJ	Real	Constraint coefficient

Created By: Module IFP

Entity: **DCONEP**

Entity Type: Relation

Description: Contains the principle strain constraint definition by specifying the identification numbers of constrained elements.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Strain constraint set identification number
ST	Real	Principle strain limit in tension
SC	Real	Principle strain limit in compression
SS	Real	Principle strain limit in shear
ETYPE	Text (8)	Element type
LAYRNUM	Integer	Layer number of a composite element
EID	Integer > 0	Element identification number

Created By: Module IFP

Entity: **DCONEPM**

Entity Type: Relation

Description: Contains the principle strain constraint definition by specifying the material identification numbers.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Strain constraint set identification number
ST	Real	Principle strain limit in tension
SC	Real	Principle strain limit in compression
SS	Real	Principle strain limit in shear
MID	Integer > 0	Material identification number

Created By: Module IFP

Entity: **DCONEPP**

Entity Type: Relation

Description: Contains the principle strain constraint definition by specifying the element property identification numbers

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Strain constraint set identification number
ST	Real	Principle strain limit in tension
SC	Real	Principle strain limit in compression
SS	Real	Principle strain limit in shear
PTYPE	Text (8)	Property type
LAYRNUM	Integer	Layer number of a composite element
PID	Integer > 0	Property identification number

Created By: Module IFP

Entity: **DCONF**

Entity Type: Relation

Description: Contains the definition of the synthetic response constraints or a synthetic objective function as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
LABEL	Text(8)	User defined label for the design function
FNAME	Text(8)	Name of the function defined in Functions packet
ARGLABEL	Text(8)	Name of the argument defined in functions packet
INTARG	Integer > 0	Integer value of the argument
RELARG	Real > 0.0	Real value of the argument

Created By: Module IFP

Entity: **DCONFLT**

Entity Type: Relation

Description: Contains the definition of the flutter constraint as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
GFACT	Real > 0.0	Constraint definition scaling factor
VTYPE	Text(8)	Text string identifying the velocity type for the table = TRUE for true velocities = EQUIV for equivalent velocities
VI	Real > 0.0	Velocity value
GAMAI	Real	Damping value

Created By: Module IFP

Notes:

1. The relation contains one tuple for each velocity, damping pair given in the Bulk Data.

Entity: **DCONFRQ**

Entity Type: Relation

Description: Contains the frequency constraint definition as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Aerodynamic set identification of the imposed constraint
MODE	Integer > 0	Mode number of the frequency to be constrained
CTYPE	Text (8)	Constraint type either UPPER or LOWER
FRQALL	Real > 0.0	The frequency constraint value

Created By: Module IFP

Entity: **DCONFT**

Entity Type: Relation

Description: Contains the fiber/transverse strain constraint definition by specifying the identification numbers of constrained elements.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Strain constraint set identification number
EFT	Real > 0.0	Tensile strain limit in the fiber direction
EFC	Real	Compressive strain limit in the fiber direction
ETT	Real > 0.0	Tensile strain limit in the transverse direction
ETC	Real	Compressive strain limit in the transverse direction
ETYPE	Text (8)	Element type
LAYRNUM	Integer	Layer number of a composite element
EID	Integer > 0	Element identification number

Created By: Module IFP

Entity: **DCONFTM**

Entity Type: Relation

Description: Contains the fiber/transverse strain constraint definition by specifying the material identification numbers.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Strain constraint set identification number
EFT	Real > 0.0	Tensile strain limit in the fiber direction
EFC	Real	Compressive strain limit in the fiber direction
ETT	Real > 0.0	Tensile strain limit in the transverse direction
ETC	Real	Compressive strain limit in in the transverse direction
MID	Integer > 0	Material identification number

Created By: Module IFP

Entity: **DCONFTP**

Entity Type: Relation

Description: Contains the fiber/transverse strain constraint definition by specifying the element property identification numbers.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Strain constraint set identification number
EFT	Real > 0.0	Tensile strain limit in the fiber direction
EFC	Real	Compressive strain limit in the fiber direction
ETT	Real > 0.0	Tensile strain limit in the transverse direction
ETC	Real	Compressive strain limit in in the transverse direction
PTYPE	Text(8)	Property type
LAYRNUM	Integer	Layer number of a composite element
PID	Integer > 0	Property identification number

Created By: Module IFP

Entity: **DCONLAM**

Entity Type: Relation

Description: Contains the laminate composition constraints as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
CTYPE	Text (8)	Constraint type, either UPPER or LOWER
PERCENT	Real	Percent allowable ply thickness
PLYNUM	Integer > 0 or -1	Ply number or -1 if PLYSET is used
PLYSET	Integer > 0 or -1	PLYLIST identification number or -1 if PLYNUM is used
LAMCHAR	Text (8)	The string ALL or blank if LAMSET is used
LAMSET	Integer = 0	PLYLIST identification number or 0 if LAMCHAR=ALL
SID	Integer > 0	ELEMLIST set identification number

Created By: Module IFP

Entity: **DCONLIST**

Entity Type: Relation

Description: Contains the definition of the constraint lists as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
CTYPE	Text (8)	Constraint type identifier
NRFAC	Real	Retention factor for minimum number of constraints
EPS	Real	Lower bound value for constraint selection by value

Created By: Module IFP

Entity: **DCONLMN**

Entity Type: Relation

Description: Contains the laminate minimum gauge constraints as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
MINTHK	Real > 0.0	Allowable minimum gauge
LAMCHAR	Text (8)	The string ALL or blank if LAMSET is used
LAMSET	Integer = 0	PLYLIST identification number or 0 if LAMCHAR=ALL
SID	Integer > 0	ELEMLIST set identification number
SIDEONLY	Integer = 1 or NULL	If 1, SIDEONLY indicates that this constraint is redundant with a side constraint.

Created By: Module IFP

Entity: **DCONPMN**

Entity Type: Relation

Description: Contains the ply minimum gauge constraints as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
MINTHK	Real > 0.0	Allowable minimum gauge
PLYNUM	Integer > 0 or -1	Ply number or -1 if PLYSET is used
PLYSET	Integer > 0 or -1	PLYLIST identification number or -1 if PLYNUM is used
SID	Integer > 0	ELEMLIST set identification number
SIDEONLY	Integer = 1 or NULL	If 1, SIDEONLY indicates that this constraint is redundant with a side constraint.

Created By: Module IFP

Entity: **DCONSCF**

Entity Type: Relation

Description: Contains the definition of a constraint on the flexible stability derivative at the reference grid point associated with the force or moment due to a trim parameter or control surface deflection of the form.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Constraint set identification
ACCLAB	Text (8)	Structural acceleration label
PRMLAB	Text (8)	Constrained control surface label or aeroelastic trim parameter
CTYPE	Text (8)	Constraint type
PRMREQ	Real	Stability coefficient bounds
UNITS	Text (8)	Stability coefficient units

Created By: Module IFP

Entity: **DCONSDE**

Entity Type: Relation

Description: Contains the definition of BAR element cross-sectional dimension side constraints.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
TMIN	Real ≥ 0.0	Minimum cross-sectional dimension in design
TMAX	Real ≥ 0.0	Maximum cross-sectional dimension in design
ETYPE	Text (8)	Element type. BAR
DVSYMBL	Text (8)	Cross-section dimension symbol (D1, D2, D3, D4, D5, D6, D7, D8, D9 or D10)
EID	Integer > 0	Element identification number

Created By: Module IFP

Entity: **DCONSDL**

Entity Type: Relation

Description: Contains the definition of BAR element cross-sectional dimension side constraints.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
TMIN	Real ≥ 0.0	Minimum cross-sectional dimension in design
TMAX	Real ≥ 0.0	Maximum cross-sectional dimension in design
DVSYMBL	Text (8)	Cross-section dimension symbol (D1, D2, D3, D4, D5, D6, D7, D8, D9 or D10)
ELID	Integer > 0	Element identification number

Created By: Module IFP

Entity: **DCONTH2**

Entity Type: Relation

Description: Contains the list of layers of composite elements for which thickness constraints are always to be retained in optimization with shape function design variable linking as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
ETYPE	Text (8)	Element type. One of the following: QUAD4 QDMEM1 TRIA3 TRMEM
PLYNUM	Integer > 0 or -1	Ply number or -1 indicating PLYSET is used
PLYSET	Integer > 0 or -1	PLYLIST set identification or -1 indicating PLYNUM is used
EID	Integer > 0	Element identification number

Created By: Module IFP

Entity: **DCONTH3**

Entity Type: Relation

Description: Contains the list of BAR element cross-sectional dimensions for which side constraints are always to be retained in optimization with shape function design variable linking as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
ETYPE	Text (8)	Element type. BAR
DVSYMBL	Text (8)	Cross-section dimension symbol (D1, D2, D3, D4, D5, D6, D7, D8, D9 or D10)
EID	Integer > 0	Element identification number

Created By: Module IFP

Entity: **DCONTHK**

Entity Type: Relation

Description: Contains the list of elements for which thickness constraints are always to be retained in optimization with shape function design variable linking as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
ETYPE	Text (8)	Element type. One of the following: BAR QUAD4 ELAS ROD MASS SHEAR QDMEM1 TRIA3 TRMEM
EID	Integer > 0	Element identification number

Created By: Module IFP

Entity: **DCONTRM**

Entity Type: Relation

Description: Contains the definitions of a trim parameter constraint.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Constraint set identification
PRMLAB	Text (8)	Constrained control surface label or aeroelastic trim parameter
CTYPE	Text (8)	Constraint type
PRMREQ	Real	Trim parameter bound

Created By: Module IFP

Entity: **DCONTW**

Entity Type: Relation

Description: Contains the Tsai-Wu stress constraint definition by specifying the identification numbers of constrained elements.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Stress constraint set identification
XT	Real > 0.0	Tensile stress limit in the longitudinal direction
XC	Real	Compressive stress limit in the longitudinal direction
YT	Real > 0.0	Tensile stress limit in the transverse direction
YC	Real	Compressive stress limit in the transverse direction
SS	Real > 0.0	Shear stress limit for in-plane stress
F12	Real	Tsai-Wu interaction term
ETYPE	Text (8)	Element type
LAYRNUM	Integer	Layer number of a composite element
EID	Integer > 0	Element identification number

Created By: Module IFP

Entity: **DCONTWM**

Entity Type: Relation

Description: Contains the Tsai-Wu stress constraint definition by specifying the material identification numbers.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Stress constraint set identification
XT	Real > 0.0	Tensile stress limit in the longitudinal direction
XC	Real	Compressive stress limit in the longitudinal direction
YT	Real > 0.0	Tensile stress limit in the transverse direction
YC	Real	Compressive stress limit in the transverse direction
SS	Real > 0.0	Shear stress limit for in-plane stress
F12	Real	Tsai-Wu interaction term
MID	Integer > 0	Material identification number

Created By: Module IFP

Entity: **DCONTWP**

Entity Type: Relation

Description: Contains the Tsai-Wu stress constraint definition by specifying the element property identification numbers.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Stress constraint set identification
XT	Real > 0.0	Tensile stress limit in the longitudinal direction
XC	Real	Compressive stress limit in the longitudinal direction
YT	Real > 0.0	Tensile stress limit in the transverse direction
YC	Real	Compressive stress limit in the transverse direction
SS	Real > 0.0	Shear stress limit for in-plane stress
F12	Real	Tsai-Wu interaction term
PTYPE	Text (8)	Property type
LAYRNUM	Integer	Layer number of a composite element
PID	Integer > 0	Property identification number

Created By: Module IFP

Entity: **DCONVM**

Entity Type: Relation

Description: Contains the Von-Mises stress constraint definition by specifying the identification numbers of constrained elements.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Stress constraint set identification number
ST	Real	Stress limit in tension
SC	Real	Stress limit in compression
SS	Real	Stress limit in shear
ETYPE	Text (8)	Element type
LAYRNUM	Integer	Layer number of a composite element
EID	Integer > 0	Element identification number

Created By: Module IFP

Entity: **DCONVMM**

Entity Type: Relation

Description: Contains the Von-Mises stress constraint definition by specifying the material identification numbers.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Stress constraint set identification number
ST	Real	Stress limit in tension
SC	Real	Stress limit in compression
SS	Real	Stress limit in shear
MID	Integer > 0	Material identification number

Created By: Module IFP

Entity: **DCONVMP**

Entity Type: Relation

Description: Contains the Von-Mises stress constraint definition by specifying the element property identification numbers.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Stress constraint set identification number
ST	Real	Stress limit in tension
SC	Real	Stress limit in compression
SS	Real	Stress limit in shear
PTYPE	Text (8)	Property type
LAYRNUM	Integer	Layer number of a composite element
PID	Integer > 0	Property identification number

Created By: Module IFP

Entity: **DDELDV**

Entity Type: Matrix

Description: Matrix of sensitivities of the trim angles to changes in the design variables.

Matrix Form: The number of rows is equal to the number of trim parameters while the number of columns is equal to the number of active flight conditions times the number of design variables.

Created By: Module AEROSENS

Notes:

1. DDELDV is needed only when the design task includes aero elastic trim and the flight conditions have been determined to be active by module ABOUND.
2. DDELDV is determined through the solution of the equation:

$$[RHS]\ [DDELDV] = [DRHS]$$

Entity: **DDMVI**

Entity Type: Unstructured

Description: Contains the nonlinear finite difference mass design sensitivity matrices.

Entity Structure:

Record:

i. Contains all or a portion of the nonlinear mass design sensitivity matrix for a given design variable.

Created By: Module NLEMA1

Notes:

1. Relation DGMMCT contains connectivity and MCODE information which defines how the matrices are stored.
2. The sensitivity matrices are stored in the same precision as the MGG matrix.

Entity: **DDPGRV**

Entity Type: Matrix

Description: Contains the nonlinear gravity load finite difference sensitivities for each gravity load set referenced in solution control.

Matrix Form: A variable-sized matrix having one row for each structural degree of freedom and one column for each gravity load condition for each design variable (including the zeroth design variable). The order of the matrix columns is:

(A) The NGRAV columns for each gravity load set for the zeroth design variable in load set id order.

(B) The NGRAV columns for each gravity load set for the first design variable in load set id order, etc.

Created By: NLLODGEN

Notes:

1. This matrix is empty if no gravity loads are referenced in solution control or in a LOAD Bulk Data entry, or if there are no nonlinear design variables.
2. This matrix and matrix [DPGRVI] constitute the total gravity load sensitivities.

Entity: **DDPTHV**

Entity Type: Matrix

Description: Contains the nonlinear thermal load finite difference sensitivities for each thermal load set referenced in the solution control.

Matrix Form: A variable-sized matrix having one row for each structural degree of freedom and one column for each thermal load condition for each design variable (including the zeroth design variable). The order of the matrix columns is:

(A) The NTHERM columns for each thermal load set for the zeroth design variable in load set id order.

(B) The NTHERM columns for each thermal load set for the first design variable in load set id order, etc.

Created By: Module NLLODGEN

Notes:

1. This matrix is empty if no thermal loads are referenced in solution control, or if there are no nonlinear design variables.
2. This matrix and matrix [DPTHVI] constitute the total thermal load sensitivities.

Entity: **DDVCT**

Entity Type: Relation

Description: Contains the data required for the assembly of the nonlinear portion (in finite difference form) of design sensitivity matrices. This relation is sorted first by DVID and then by KSIL.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DVID	Integer > 0	Design variable identification number
PREF	Real	Design variable linking factor
ALPHA	Real	Exponential power associated with the design variable
KSIL	Integer > 0	Internal identification for a grid connected to the element
KCODE	Integer > 0	A code word denoting the form in which the element stiffness matrix is stored
MCODE	Integer > 0	A code word denoting the form in which the element mass matrix is stored
TCODE	Integer ≥ 0	A code word denoting the form in which the element thermal loads sensitivities are stored
TREFPT	Integer	The position in TREF for the associated reference temperature
NODES	Integer > 0	The number of nodes connected to the element
IREC	Integer > 0	The record number of the unstructured entity KELM, MELM, or TELM that contains the partition of the element matrix
ASILS	Ivector (4)	List of associated sils of the element in sorted order

Created By: Module NLEMG

Notes:

1. This relation contains one tuple for each nonlinear design variable for each node of each structural element.
2. The code words KCODE, MCODE and TCODE have the following definition:

KCODE/MCODE/TCODE	MEANING
7	Element has extensional DOF's only
8	Element has rotational DOF's only
9	Element has both extension and rotation
10	Element matrix has only diagonal extensional entries
11	Element matrix has only diagonal rotational entries
12	Element matrix has diagonal entries for all grid point DOF's

3. A KCODE, MCODE or TCODE of zero implies that the element has no associated stiffness, mass or thermal load.
4. Design variable offset value is stored in INFO(11) from NLEMG. It is used for the pseudo design variable spawned to handle the non-linear portion of the BAR element stiffness.
5. 7, 8, and 9 are the only values supported for TCODE.
6. This relation contains no tuples for linear design variables.

Entity: **DDVSIZE**

Entity Type: Unstructured

Description: Contains memory allocation information on the DDVCT relation.

Entity Structure:

Record 1.

WORD 1	Maximum number of DDVCT tuples associated with any one design variable other than zero
WORD 2	Number of tuples connected to "design variable" zero
WORD 3 through NDV+2	Number of tuples connected to each design variable

Created By: Module NLEMG

Notes:

1. Entity contains one record with NDV +2 words.

Entity: **DELB**

Entity Type: Matrix

Description: Matrix containing trim parameters used as initial conditions in the nuclear blast response calculations.

Matrix Form: Real matrix with three rows and one column.

Created By: MAPOL

Entity: **DELM**

Entity Type: Matrix

Description: Matrix containing trim that precedes a nuclear blast response calculation.

Matrix Form: Rectangular matrix with two rows and three columns.

Created By: MAPOL

Entity: **DELTA**

Entity Type: Subscripted Matrix

Description: A vector of trim parameters for each flight condition.

Matrix Form: The number of rows is dependent on the type of trim analysis being performed. The number of columns is equal to the number of load conditions being applied for the current Mach number and boundary condition.

Created By: Module SAERO

Notes:

1. For symmetric analyses, there are two to four rows in DELTA, depending on the value of TRMTYP on the TRIM Bulk Data entry.

Entity: **DENSLIST**

Entity Type: Relation

Description: Contains the list of density values as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer	Set identification number
DENSITY	Real > 0.0	Density value

Created By: Module IFP.

Entity: **DESELM**

Entity Type: Relation

Description: Contains design variable connection information uniquely associating one design variable to one element.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DVID	Integer > 0	Design variable id
EID1	Integer > 0	Element identification
ETYPE1	Text (8)	Element type
VMIN	Real	Minimum value of design variable
VMAX	Real	Maximum value of design variable
VALUE	Real	Initial value of design variable
LAYERNUM	Integer	Layer of a composite material
DVSYMBL	Text(8)	Designed dimension symbol (D1, D2, D3, D4, D5, D6, D7, D8, D9, D10 for BAR element cross section dimension; A for element area, T for element thickness; M for element mass; K for element stiffness.)
LABEL	Text (8)	User label

Created By: Module IFP

Notes:

1. The LAYERNUM entry identifies the layer on the PCOMP entry for the element defined by EID1 and ETYPE1.

Entity: **DESHIST**

Entity Type: Relation

Description: Contains information on the results of major iterations in the design task.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
NITER	Integer > 0	Iteration number for optimization
OBJEXACT	Real	Exact objective function value
OBJAPROX	Real	Approximate objective function value after the **NITER**th approximate problem has been solved.
NFUNC	Integer ≥ 0	Number of function evaluations in the current iteration
NGRAD	Integer ≥ 0	Number of gradient evaluations in the current iterations
NCON	Integer ≥ 0	Number of constraints
NAC	Integer ≥ 0	Number of active constraints
NVC	Integer ≥ 0	Number of violated constraints
NLBS	Integer ≥ 0	Number of active lower bound side constraints
NUBS	Integer ≥ 0	Number of active upper bound side constraints
CONVRGD	Integer ≥ 0	Convergence flag

Created By: Module DESIGN

Notes:

1. The CONVRGD parameter has the following definition:

CONVRGD	MEANING
0	Design has not converged
1	Design has converged

Entity: **DESLINK**

Entity Type: Relation

Description: Contains design variable linking information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0	Element identification number
ETYPE	Text (8)	Element type. One of the following: BAR QDMEM1 ELAS QUAD4 ROD SHEAR TRIA3 TRMEM
LAYRNUM	Integer = 0	Layer number, = 0 if noncomposite element
DVSYMBL	Text(8)	Designed dimension symbol (D1, D2, D3, D4, D5, D6, D7, D8, D9, D10 for BAR element cross section dimension; A for element area, T for element thickness; M for element mass; K for element stiffness.)
DVID	Integer > 0	Global design variable connected to this EID/LAYER
PREF	Real	Design Variable Linking Factor ([PTRANS] matrix term)

Created By: MAKEST

Notes:

1. There is one entry for each local design variable for each global design variable linked to it. Basically, this is a relational form of the [PTRANS] matrix.

Entity: **DESVARP**

Entity Type: Relation

Description: Contains the properties of each physically linked design variable.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DVID	Integer > 0	Design variable id
LINKID	Integer > 0	ELIST or PLIST identification number
VMIN	Real	Minimum value of the design variable
VMAX	Real	Maximum value of the design variable
VALUE	Real	Initial value of the design variable
LAYERNUM	Integer	Layer number for a composite element
LAYRLST	Integer	PLYLIST identification number for layer list
LABEL	Text (8)	User label to describe the design

Created By: Module IFP

Notes:

1. The LAYERNUM entry identifies the single ply of a composite element. LAYERNUM = -1 if LAYRLST is used.
2. The LAYRLST entry identifies the list of plies linked to the design variable. LAYRLST = -1 if LAYRNUM is used.

Entity: **DESVARS**

Entity Type: Relation

Description: Contains the properties of shape function linked design variable.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DVID	Integer > 0	Design variable id
SHAPEID	Integer > 0	SHAPE set identification number
VMIN	Real	Minimum value of the design variable
VMAX	Real	Maximum value of the design variable
VALUE	Real	Initial value of the design variable
LAYERNUM	Integer	Layer number for a composite element
LAYRLST	Integer	PLYLIST identification number for layer list
LABEL	Text (8)	User label

Created By: Module IFP

Notes:

1. The LAYERNUM entry identifies the single ply of a composite element. LAYERNUM = -1 if LAYRLST is used.
2. The LAYRLST entry identifies the list of plies linked to the design variable. LAYRLST = -1 if LAYRNUM is used.

Entity: **DFDU**

Entity Type: Matrix

Description: See Notes.

Matrix Form: A variable sized matrix having one row for each structural degree of freedom and one column for each currently active constraint.

The order of the DFDU columns is as follows for each active boundary condition:

(A) The sensitivities of active displacement constraints for each active load condition.

(B) The sensitivities of each active stress or strain constraint in each active load condition.

Created By: Module MAKDFU or MAPOL

Notes:

1. For the Gradient Method, contains the sensitivities of the currently active constraints to the global displacements for those constraints that are functions of the displacements.
2. For the Virtual Load Method, contains the sum of the sensitivity of the design dependent loads and the product of the design sensitivity stiffness matrix and the active displacement vectors.
3. The MAPOL sequence supports the following partitions of the DFPU matrix (see Theoretical Manual for the explicity formation of these submatrices):

$$DFDU \rightarrow \begin{bmatrix} \varphi \\ DFDUN \end{bmatrix}$$

$$DFDUN \rightarrow \begin{bmatrix} \varphi \\ DFDUF \end{bmatrix}$$

Entity: **DFDUF**

Entity Type: Matrix

Description: A partition of matrix DFDUN (see DFDU).

Entity: **DFDU**

Entity Type: Matrix

Description: A partition of matrix DFDUN (see DFDU).

Entity:	**DFSV**
Entity Type:	Matrix
Description:	Contains design variable nonlinear S-matrix derivatives related active stress/strain constraint sensitivity terms.
Matrix Form:	A variable sized matrix having one row for each active subcase for each design variable and one column for each currently active constraint. The order of the DFSV columns is as follows for each active boundary condition: (A) Null columns corresponding to the sensitivities of active displacement constraints for each active load condition. (B) The sensitivities of each active stress and/or strain constraint in each active load condition.
Created By:	Module MKDFSV

Entity: **DGMMCT**

Entity Type: Relation

Description: Contains data required to interpret the DDMVI unstructured entity for the purpose of generating all of the mass stiffness matrix partitions. The relation is sorted by DVID and KSIL.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DVID	Integer > 0	Design variable identification
KSIL	Integer > 0	Internal id of grid or scalar point connected to the design variable
MCODE	Integer > 0	Codeword denoting the form in which the DDMVI data are stored
NODES	Integer 0 < NODES ≤ 20	Number of nodes being processed with this KSIL
IREC	Integer > 0	Record number of DDMVI entity where data are stored
ASIL	Ivector (20)	List of associated SILS
ALPHA	Real	Exponential power associated with BAR element design variable

Created By: Module NLEMA1

Notes:

1. MCODE definitions:
   - = 1 Multiple associated grids with both extensional and rotational degrees of freedom.
   - = 2 Column being assembled is a scalar point. Associated SILS may or may not be scalar points.
   - = 3 Column being assembled is a grid point. At least one row is a scalar point.
   - = 4 Only extensional degrees of freedom are included in the assembly process.
2. The INFO array for this entity contains additional memory management allocation data:
   - INFO(11) The number of tuples connected to the zeroth design variable.
   - INFO(12) The maximum number of tuples connected to any one of the remaining design variables.

Entity: **DISPDRVA**

Entity Type: Relation

Description: Contains the user function requested displacement response sensitivity information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
MXCOL	Integer > 0	Matrix column number in [MXDSPDVA] for the sensitivity values

Created By: Module MKAMAT

Entity: **DISPRESP**

Entity Type: Relation

Description: Contains the user function requested displacement response values.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
VALUE	Real	Response value

Created By: Module DCEVAL

Entity: **DKUG**

Entity Type: Matrix

Description: The product of the design sensitivity matrices and the active displacement vectors.

Matrix Form: The number of columns is equal to NAC, the number of active subcases times NDV, the number of design variables. The number of rows is equal to the number of terms in the g-set.

Created By: MAKDVU

Notes:

1. The sensitivity to the first design variable for all the active subcases occupies the first NAC columns. This is followed by columns for each of the remaining design variables turn.
2. The negative of the product is created in order to simplify later matrix operations.

Entity: **DKELM**

Entity Type: Unstructured

Description: Contains the element nonlinear finite difference stiffness sensitivity matrix partitions.

Entity Structure:

Record:

i the record contains a partition of the stiffness matrix with either 1, 3, or 6 entries for each node

KCODE	FORMAT OF RECORD
7	3 columns of 3 entries/node
8	3 columns of 3 entries/node
9	6 columns of 6 entries/node

Created By: Module NLEMG

Notes:

1. This entity contains one record for each *strip* of each element nonlinear design stiffness derivative matrix. A strip or partition is defined as all those columns of the element matrix associated with a pivot sil (the ID of the first DOF of a grid point or the id of a scalar point).
2. Refer to the DDVCT relation for further details, as these two data base entities are closely linked.
3. The matrix partitions are stored in the same precision as the KGG matrix.
4. DKELM and KELM are used to generate all of the element stiffness sensitivity matrix partitions.

Entity: **DKVI**

Entity Type: Unstructured

Description: Contains the stiffness design sensitivity matrices.

Entity Structure:

Record:

1. Contains all of a portion of the stiffness design sensitivity matrix for a given design variable.

Created By: Module NLEMA1

Notes:

1. Relation GMKCT contains connectivity and KCODE information which defines how the matrices are stored.
2. The sensitivity matrices are stored in the same precision as the KGG matrix.
3. DKVI is built from DKVI0 (which contains the linear stiffness design sensitivity matrix) and from the finite difference nonlinear stiffness design sensitivity.

Entity: **DKVI0**

Entity Type: Unstructured

Description: Contains the linear stiffness design sensitivity matrices.

Entity Structure:

Record:

1. Contains all of a portion of the linear stiffness design sensitivity matrix for a given design variable.

Created By: Module NLEMA1

Notes:

1. Relation GMKCT0 contains connectivity and KCODE information which defines how the matrices are stored.
2. The sensitivity matrices are stored in the same precision as the KGG matrix.
3. The INFO array contains information on the generation of the mass matrix.
   INFO(11) = 1 Generate the global mass matrix in the final analysis
   = 0 Don't generate the global mass matrix
   INFO(12) = 1 Generate the global mass matrix for the optimization

Entity: **DKVIG**

Entity Type: Unstructured

Description: Contains the stiffness matrix partitions.

Entity Structure:

Record:

1. Contains all of a portion of the stiffness matrix for a given design variable.

Created By: Module NLEMA1

Notes:

1. Relation GMKCTG contains connectivity and KCODE information which defines how the matrices are stored.
2. The matrices are stored in the same precision as the KGG matrix.
3. The INFO array contains information on the generation of the mass matrix.
   INFO(11) = 1 Generate the global mass matrix in the final analysis
   = 0 Don't generate the global mass matrix
   INFO(12) = 1 Generate the global mass matrix for the optimization

Entity: **DK1V**

Entity Type: Matrix

Description: An intermediate matrix in the calculation of the sensitivities of static aeroelastic displacements.

Matrix Form: Rectangular real matrix with the number of rows equal to the number of a-set degrees and the number of columns equal to the number of active displacement vectors times the number of design variables.

Created By: MAPOL

Notes:

1. This matrix is the solution to:

   [K11] [DK1V] = [DP1]

Entity: **DLAGS**

Entity Type: Relation

Description: Contains loading information for a dynamics load set as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DSID	Integer > 0	DLAGS set identification number
LSID	Integer > 0	Static load set id
TAU	Real	Time delay value
PHASE	Real	Phase lag value

Created By: Module IFP

Entity: **DLOAD**

Entity Type: Relation

Description: Contains dynamic loads information as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer > 0	Load set identification number
SCAL	Real	Overall scale factor
SCALI	Real	Scale factor for this tuple
LOADI	Integer > 0	ID of the associated TLOADi or RLOADi set for this tuple

Created By: Module IFP

Notes:

1. The relation is used in the transient response and/or the frequency response module.

Entity: **DLONLY**

Entity Type: Relation

Description: Contains loads information for dynamic response as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DSID	Integer > 0	DLONLY set identification number
POINT	Integer > 0	Grid, scalar or extra point ID
COMP	Integer ≥ 0	Component number
AVAL	Real	Load value

Created By: Module IFP

Notes:

1. Subroutine PREDOL processes DLONLY data and write them to the UDLOLY entity.
2. COMP is 1-6 for grid points and zero for extra or scalar points.

Entity: **DMAG**

Entity Type: Matrix

Description: Matrix product of mass design sensitivity matrices and active acceleration vectors.

Matrix Form: The number of columns is equal to NAC, the number of active subcases, times NDV, the number of design variables. The number of rows is equal to the number g-set degrees of freedom.

Created By: MAKDVU

Entity: **DMELM**

Entity Type: Unstructured

Description: An unstructured database entity that contains the element nonlinear finite difference mass sensitivity matrix partitions.

Entity Structure:

Record: the record contains a partition of the stiffness matrix with either one, three, or six entries for each node

KCODE	FORMAT OF RECORD
7	3 columns of 3 entries/node
10	3 columns of 1 entry/node (diagonal)

Created By: Module NLEMG

Notes:

1. This entity contains one record for each *strip* of each element nonlinear design mass sensitivity matrix. A strip or partition is defined as all those columns of the element matrix associated with a pivot sil (the id of the first dof of a grid point or the id of a scalar point).
2. Refer to the DDVCT relation for further details, as these two database entities are closely linked.
3. The matrix partitions are stored in the same precision as the MGG matrix.
4. DMELM and MELM are used to generate all of the element mass sensitivity matrix partitions.

Entity: **DMIG**

Entity Type: Relation

Description: Contains the direct matrix input data for structural matrices as defined in the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
NAME	Text 8	Matrix entity name
PREC	Text 4	Matrix precision
FORM	Text 8	Matrix form
GCOL	Integer	External point identification of column index
CCOL	Integer	Grid component number of column index
GROW	Integer	External point id of row index
CROW	Integer	Grid point component of row index
XIJ	Real	Real part of matrix term
YIJ	Real	Imaginary part of matrix term

Created By: Module IFP

Entity: **DMU**

Entity Type: Matrix

Description: The reduced mass sensitivity matrix used in the calculation of sensitivities of displacements when there are unrestrained degrees of freedom.

Matrix Form: The number of rows is equal to the number of SUPPORT degrees of freedom and the number of columns is equal to the number of columns in DMUG.

Created By: MAPOL

Notes:

1. This matrix is computed from:

$$[\ DMU\ ] = [\ D\ ]^{T} * [\ DMUL\ ] + [\ DMUR\ ]$$

Entity: **DMUA**

Entity Type: Matrix

Description: A partition of the DMUF matrix (see DMUG).

Entity: **DMUF**

Entity Type: Matrix

Description: A partition of the DMU matrix (see DMUG).

Entity: **DMUG**

Entity Type: Matrix

Description: Contains the product of the mass design sensitivity matrices and the active acceleration vectors.

Matrix Form: The number of columns is equal to NAC, the number of active subcases times NDV, the number of design variables. The number of rows is equal to the number of degrees of freedom in the g-set.

Created By: Module MAKDVU

Notes:

1. This matrix is created only when there are unrestrained degrees of freedom.
2. The sensitivity to the first design variable for all the active subcases occupies the first NAC columns. This is followed by columns for each of the remaining design variables in turn.
3. The negative of the product is created in order to simplify the later matrix operations.
4. The MAPOL sequence supports the partitions of the DMUG matrix (see the Theoretical Manual for the explicity formation of these submatrices:)

$$DMUG \rightarrow \begin{bmatrix} \varphi \\ DMUN \end{bmatrix}$$

$$DMUN \rightarrow \begin{bmatrix} \varphi \\ DMUF \end{bmatrix}$$

$$DMUF \rightarrow \begin{bmatrix} DMUO* \\ DMUA \end{bmatrix}$$

$$DMUA \rightarrow \begin{bmatrix} DMUR \\ DMUL \end{bmatrix}$$

* Generated for Guyan reduction only.

Entity: **DMUL**

Entity Type: Matrix

Description: A partition of the DMUA matrix (see DMUG).

Entity: **DMUN**

Entity Type: Matrix

Description: A partition of the DMUG matrix (see DMUG).

Entity: **DMUO**

Entity Type: Matrix

Description: A partition of the DMUF matrix (see DMUG).

Entity: **DMUR**

Entity Type: Matrix

Description: A partition of the DMUA matrix (see DMUG).

Entity: **DMVI**

Entity Type: Unstructured

Description: Contains the mass design sensitivity matrices.

Entity Structure:

Record:

i. Contains all or a portion of the mass design sensitivity matrix for a given design variable.

Created By: Module NLEMA1

Notes:

1. Relation GMMCT contains connectivity and MCODE information which defines how the matrices are stored.
2. The sensitivity matrices are stored in the same precision as the MGG matrix.
3. DMVI is built from DMVI0 (which contains the linear mass design sensitivity matrix) and from the finite difference nonlinear mass design sensitivity.

Entity: **DMVI0**

Entity Type: Unstructured

Description: Contains the linear mass design sensitivity matrices.

Entity Structure:

Record:

i. Contains all or a portion of the linear mass design sensitivity matrix for a given design variable.

Created By: Module NLEMA1

Notes:

1. Relation GMMCT0 contains connectivity and MCODE information which defines how the matrices are stored.
2. The sensitivity matrices are stored in the same precision as the MGG matrix.

Entity: **DMVID**

Entity Type: Unstructured

Description: Contains the nonlinear design mass matrix partitions.

Entity Structure:

Record:

i. Contains all or a portion of the nonlinear design mass matrix for the zeroth design variable.

Created By: Module NLEMA1

Notes:

1. Relation GMMCTD contains connectivity and MCODE information which defines how the matrices are stored.
2. The matrices are stored in the same precision as the MGG matrix.

Entity: **DMVIG**

Entity Type: Unstructured

Description: Contains the mass matrix partitions.

Entity Structure:

Record:

i. Contains all or a portion of the mass matrix for a given design variable.

Created By: Module NLEMA1

Notes:

1. Relation GMMCTG contains connectivity and MCODE information which defines how the matrices are stored.
2. The matrices are stored in the same precision as the MGG matrix.

Entity: **DPAV**

Entity Type: Matrix

Description: Partition of the DPFV matrix (see DPGV).

Entity: **DPFV**

Entity Type: Matrix

Description: Partition of the DPNV matrix (see DPGV).

Entity: **DPGRVD**

Entity Type: Matrix

Description: Contains the gravity loads related to nonlinear design elements for each gravity load set referenced in solution control.

Matrix Form: A variable-sized matrix having one row for each structural degree of freedom and one column for each gravity load condition for each design variable (including the zeroth design variable). The order of the matrix columns is:

(A) The NGRAV columns for each gravity load set for the zeroth design variable in load set id order.

(B) The NGRAV columns for each gravity load set for the first design variable in load set id order, etc.

Created By: NLLODGEN

Notes:

1. This matrix is empty if no gravity loads are referenced in solution control or in a LOAD Bulk Data entry, or if there are no nonlinear design variables.
2. [DPGRVD] and [DDPGRI] are used to generate the total gravity loads.

Entity: **DPGRVI**

Entity Type: Matrix

Description: Contains the linear gravity load sensitivities for each gravity load set referenced in solution control.

Matrix Form: A variable-sized matrix having one row for each structural degree of freedom and one column for each gravity load condition for each design variable (including the zeroth design variable). The order of the matrix columns is:

(A) The NGRAV columns for each gravity load set for the zeroth design variable in load set id order.

(B) The NGRAV columns for each gravity load set for the first design variable in load set id order, etc.

Created By: LODGEN

Notes:

1. This matrix is empty if no gravity loads are referenced in solution control or in a LOAD Bulk Data entry, or if there are neither linear design variables nor nondesigned elements.
2. [DPGRVI] and [DDPGRV] constitute the total gravity load sensitivities.
3. [DPGRVI] and [DPGRVD] are used to generate the total gravity load at the current design point.

Entity: **DPGV**

Entity Type: Matrix

Description: See Notes.

Matrix Form: Real rectangular matrix with one row for each g-set degree of freedom. The number of columns is equal to the number of active subcases times the number of design variables.

Created By: MAPOL or MAKDFU

Notes:

1. For the Gradient Method, contains the right-hand sides for the sensitivity calculations. If there are design dependent loads, DPGV is the sum of DPVJ and DKUG. If there are no design dependent loads, DPGV is equivalent to DKUG.
2. For the Virtual Load Method, contains the sensitivities of the currently active constraints to the global displacements.
3. The MAPOL sequence supports the following partitions of the DPVG matrix (see the Theoretical Manual for the explicit formation of these matrices):

$$DPGV \rightarrow \begin{bmatrix} \varphi \\ DPNV \end{bmatrix}$$

$$DPNV \rightarrow \begin{bmatrix} \varphi \\ DPFV \end{bmatrix}$$

$$DPFV \rightarrow \begin{bmatrix} DPOV^{*} \\ DPAV \end{bmatrix}$$

$$DPAV \rightarrow \begin{bmatrix} DPRV \\ DPLV \end{bmatrix}$$

* Generated for the Guyan reduction only.

Entity: **DPLV3**

Entity Type: Matrix

Description: Partition of the DPAV matrix (see DPGV).

Entity: **DPNV**

Entity Type: Matrix

Description: Partition of the DPGV matrix (see DPGV).

Entity: **DPOV**

Entity Type: Matrix

Description: Partition of the DPFV matrix (see DPGV).

Entity: **DPRV**

Entity Type: Matrix

Description: Partition of the DPFV matrix (see DPGV).

Entity: **DPTHVD**

Entity Type: Matrix

Description: Contains the thermal loads related to nonlinear design elements for each thermal load set referenced in the solution control.

Matrix Form: A variable-sized matrix having one row for each structural degree of freedom and one column for each thermal load condition for each design variable (including the zeroth design variable). The order of the matrix columns is:

(A) The NTHERM columns for each thermal load set for the zeroth design variable in load set id order.

(B) The NTHERM columns for each thermal load set for the first design variable in load set id order, etc.

Created By: Module NLODGEN

Notes:

1. This matrix is empty if no thermal loads are referenced in solution control, or if there are no nonlinear design variables.
3. [DPTHVD] and [DPTHVI] are used to generate the total thermal loads.

Entity: **DPTHVI**

Entity Type: Matrix

Description: Contains the linear thermal loads sensitivities for each thermal load set referenced in the solution control.

Matrix Form: A variable-sized matrix having one row for each structural degree of freedom and one column for each thermal load condition for each design variable (including the zeroth design variable). The order of the matrix columns is:

(A) The NTHERM columns for each thermal load set for the zeroth design variable in load set id order.

(B) The NTHERM columns for each thermal load set for the first design variable in load set id order, etc.

Created By: Module LODGEN

Notes:

1. This matrix is empty if no thermal loads are referenced in solution control, or if there are neither linear design variables and nor nondesigned element.
2. [DPTHVI] and [DDPTHV] constitute the total thermal loads sensitivities.
3. [DPTHVI] and [DPTHVD] are used to generate the total thermal load.

Entity: **DPVJ**

Entity Type: Matrix

Description: Contains the sensitivities of the active loads to the design variables.

Matrix Form: A variable-size matrix having one row for each structural degree of freedom and one column for each active load in the current active boundary condition. The order of the columns is as follows:

(A) The sensitivities of each active load condition in load condition order for the first design variable.

(B) The sensitivities of each active load condition in load condition order to the second design variable etc

Created By: Module DDLOAD

Notes:

1. If any <u>one</u> load condition in the current active boundary condition is design dependent, the full DPVJ matrix must be created so that the DPVJ and the DKUG matrices are conformable.
2. If no design depend loads exist in the current active boundary condition, the matrix is empty.
3. The DPVJ is currently built from the appropriate linear combinations of DPTHVI and DPGRVI columns.

Entity: **DP1**

Entity Type: Matrix

Description: A load sensitivity matrix used in the calculation of displacement sensitivities when there are unrestrained degrees of freedom.

Matrix Form: A rectangular matrix with the number of rows equal to the number of degrees of freedom in the a-set and the number of columns is equal to the product of the number of columns equal the number of columns in DPGV.

Created By: MAPOL

Notes:

1. DP1 is computed by performing a ROWMERGE on matrix entities DMU and DPGL.

Entity: **DRHS**

Entity Type: Matrix

Description: Sensitivity of the applied loads to the changes in the design variables after they have been reduced to the support set.

Matrix Form: The number of rows is equal to the number of degrees of freedom in the r-set while the number of design variables times the number of active load cases as determined by the ABOUND module.

Created By: MAPOL

Notes:

1. If an inertia relief sensitivity analysis is being per formed, DRHS is DPRV plus the transpose of D times DPLV.
2. If a static aeroelastic sensitivity analysis is being performed, K21 times DKLV is subtracted from the DRHS defined above.

Entity: **DTELM**

Entity Type: Unstructured

Description: Contains the element design nonlinear thermal load sensitivity partitions for nonlinear designed elements if any thermal loads have been defined in the model.

Entity Structure:

Record:

i. Each record contains the geometric and material thermal load sensitivity partitions for each nonlinear design element in the model if any thermal loads have been defined in the model.

Created By: Module NLEMG

Notes:

1. This entity contains one record for each partition of each nonlinear element thermal load sensitivity matrix. A partition is that portion of the matrix connected to one pivot sil.
2. Refer to the DDVCT relation documentation for further details.
3. The DTELM terms are stored in the same precision as the PG matrix.
4. DTELM and TELM are used to generate all of the element thermal load sensitivity matrix partitions.

Entity: **DTSLP**

Entity Type: Matrix

Description: A matrix used in the nuclear blast calculation to compute the slopes at the aerodynamic panels given the modal participation factors.

Matrix Form: A real, rectangular matrix with the number of rows equal to the number of aerodynamic panels and the number of columns equal to the number of retained modes.

Created By: MAPOL

Notes:

1. DTSLP is computed using:

$$[DTSLP] = [BLSTJA]^{T} [PHIB]$$

Entity: **DUAD**

Entity Type: Matrix

Description: Matrix of sensitivities of the a-set accelerations to changes in the design variables.

Matrix Form: The number of rows is equal to the number of degrees of freedom in the a-set while the number of columns is equal to the number of active load cases times the number of design variables.

Created By: MAPOL

Notes:

1. This matrix is formed by merging DURD and DULD.
2. This matrix is constructed only when there is inertia relief and when the load vectors have been determined to be active by module ABOUND.

Entity: **DUAV**

Entity Type: Matrix

Description: Sensitivity of displacements in the a-set.

Matrix Form: The number of columns is equal to the number of active subcases times the number of design variables. The number of rows is equal to the number of terms in the a-set.

Created By: MAPOL, Module AEROSENS or Module FBS

Notes:

1. For static analysis without inertia relief DUAV is determined by FBS; for inertia relief, DUAV is merged from DURV and DULV. For static aeroelasticity, DUAV is calculated in AEROSENS.

Entity: **DUFV**

Entity Type: Matrix

Description: Sensitivity of displacements in the f-set.

Matrix Form: The number of columns is equal to the number of active subcases times the number of design variables. The number of rows is equal to the number of terms in the f-set.

Created By: MAPOL

Notes:

1. For generalized dynamic reduction, DUFV is obtained from DUAV and GSUBO. For Guyan Reduction, DUFV is obtained from merging DUAV and temporary matrix UO which represents the sensitivity of the displacements in the o-set.

Entity: **DUG**

Entity Type: Matrix

Description: Summation of the DKUG and DMUG matrices.

Matrix Form: The number of columns is equal to NAC, the number of active subcases, times NDV, the number of design variables. The number of columns is equal to the number of degrees of freedom in the g-set.

Created By: MAPOL

Notes:

1. If there are no SUPPORT degrees of freedom, DUG is equivalenced into DKUG.

Entity: **DULD**

Entity Type: Matrix

Description: Matrix of sensitivities of the l-set accelerations to changes in the design variables.

Matrix Form: The number of rows is equal to the number of degrees of freedom in the l-set while the number of columns is equal to the number of active load cases times the number of design variables.

Created By: MAPOL

Notes:

1. This matrix is formed by multiplying D by DURD.
2. This matrix is constructed only when there is inertia relief and when the load vectors have been determined to be active by module ABOUND.

Entity: **DULV**

Entity Type: Matrix

Description: Sensitivity of displacements in the l-set. The computed sensitivity of the active subcases to changes in the design variable.

Matrix Form: The number of columns is equal to the number of active subcases times the number of design variables. The number of rows is equal to the number of terms in the l-set.

Created By: Module FBS

Notes:

1. This matrix is created only when there is statics with inertia relief and when the load vectors have been determined to be active by module ABOUND.

Entity: **DURD**

Entity Type: Matrix

Description: The sensitivity of the rigid body acceleration matrix to changes in the design variables.

Matrix Form: Real and rectangular. The number of rows is equal to the number of degrees of freedom in the r-set while the number of columns is equal to the number of active subcases times the number of design variables.

Created By: Module INERTIA

Notes:

1. This matrix is formed only when there is inertia relief and the applied load has been determined to be active by module ABOUND.
2. The matrix is formed by solving

$$[MRR\ ]\ [DURD\ ] = [DRHS\ ]$$

Entity: **DVCT**

Entity Type: Relation

Description: Contains the data required for the assembly of the linear portion (in analytical form) of design sensitivity matrices. Relation is sorted first by DVID and then by KSIL.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DVID	Integer > 0	Design variable identification number
PREF	Real	Design variable linking factor
ALPHA	Real	Exponential power associated with the design variable
KSIL	Integer > 0	Internal identification for a grid connected to the element
KCODE	Integer > 0	A code word denoting the form in which the element stiffness matrix is stored
MCODE	Integer > 0	A code word denoting the form in which the element mass matrix is stored
TCODE	Integer ≥ 0	A code word denoting the form in which the element thermal loads sensitivities are stored
TREFPT	Integer	The position in TREF for the associated reference temperature
NODES	Integer > 0	The number of nodes connected to the element
IREC	Integer > 0	The record number of the unstructured entity KELM, MELM, or TELM that contains the partition of the element matrix
ASILS	Ivector (32)	List of associated sils of the element in sorted order

Created By: Module EMG

Notes:

1. This relation contains one tuple for each design variable for each node of each structural element.
2. The words KCODE, MCODE and TCODE have the following definition:

KCODE/MCODE/TCODE FOR SCALAR ELEMENTS	MEANING (No meaning for TCODE)
1	Scalar point connected to ground
2	Grid point component connected to a scalar point
3	Scalar point connected to grid component
4	Scalar point connected to a scalar point
5	Grid point component connected to ground
6	Grid point component connected to a second grid point component

KCODE/MCODE/TCODE FOR OTHER ELEMENTS (CONNECTED TO GRID POINTS)	MEANING
7	Element has extensional DOF's only
8	Element has rotational DOF's only
9	Element has both extension and rotation
10	Element matrix has only diagonal extensional entries
11	Element matrix has only diagonal rotational entries
12	Element matrix has diagonal entries for all grid point DOF's

3. A KCODE, MCODE or TCODE of zero implies that the element has no associated stiffness, mass or thermal load.
4. Design variable offset value is stored in INFO(11) from EMG. It is used for the pseudo design variable spawned to handle the nonlinear portion of the BAR element stiffness.
5. 7, 8, and 9 are the only values supported for TCODE.
6. This relation contains tuples for nonlinear design variables for which related records of unstructured entities KECM, MECM or TECM are empty.

Entity: **DVCTD**

Entity Type: Relation

Description: Contains the data required for the assembly of the nonlinear design element sensitivity matrices. Relation is sorted by KSIL.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DVID	Integer > 0	Design variable identification number
PREF	Real	Design variable linking factor
ALPHA	Real	Exponential power associated with the design variable
KSIL	Integer > 0	Internal identification for a grid connected to the element
KCODE	Integer > 0	A code word denoting the form in which the element stiffness matrix is stored
MCODE	Integer > 0	A code word denoting the form in which the element mass matrix is stored
TCODE	Integer ≥ 0	A code word denoting the form in which the element thermal loads sensitivities are stored
TREFPT	Integer	The position in TREF for the associated reference temperature
NODES	Integer > 0	The number of nodes connected to the element
IREC	Integer > 0	The record number of the unstructured entity KELM, MELM, or TELM that contains the partition of the element matrix
ASILS	Ivector (4)	List of associated sils of the element in sorted order

Created By: Module NLEMG

Notes:

1. This relation contains one tuple for the zeroth design variable for each node of each nonlinear design structural element.
2. The words KCODE, MCODE and TCODE have the following definition:

KCODE/MCODE/TCODE	MEANING
7	Element has extensional DOF's only
8	Element has rotational DOF's only
9	Element has both extension and rotation
10	Element matrix has only diagonal extensional entries
11	Element matrix has only diagonal rotational entries
12	Element matrix has diagonal entries for all grid point DOF's

3. A KCODE, MCODE or TCODE of zero implies that the element has no associated stiffness, mass or thermal load.
4. Design variable offset value is stored in INFO(11) from NLEMG. It is used for the pseudo design variable spawned to handle the non-linear portion of the BAR element stiffness.
5. 7, 8, and 9 are the only values supported for TCODE.
6. This relation contains no tuple for linear design variables.

Entity: **DVSIZE**

Entity Type: Unstructured

Description: Contains memory allocation information on the DVCT relation.

Entity Structure:

Record 1.

WORD 1	Maximum number of DVCT tuples associated with any one design variable other than zero
WORD 2	Number of tuples connected to "design variable" zero
WORD 3 through NDV+2	Number of tuples connected to each design variable

Created By: Module EMG

Notes:

1. Entity contains one record with NDV +2 words.

Entity: **DVSIZED**

Entity Type: Unstructured

Description: Contains memory allocation information on the DVCTD relation.

Entity Structure:

Record 1.

WORD 1	0
WORD 2	Number of tuples connected to "design variable" zero
WORD 3 through NDV+2	0

Created By: Module NLEMG

Notes:

1. Entity contains one record with NDV +2 words.

Entity: **DVTOPTE**

Entity Type: Relation

Description: Contains the designed element thickness variation type definition by specifying the element identification numbers.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
TYPE	Text (8)	Designed element thickness variation
TYPE	Text (8)	Element type
EID	Integer > 0	Element identification number

Created By: Module IFP

Entity: **DVTOPTL**

Entity Type: Relation

Description: Contains the designed element thickness variation type definition by specifying the element list set ID numbers.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
TYPE	Text (8)	Designed element thickness variation
TYPE	Text (8)	Element type
ELID	Integer > 0	Element list set ID number

Created By: Module IFP

Entity: **DVTOPTP**

Entity Type: Relation

Description: Contains the designed element thickness variation type definition by specifying the element property identification numbers.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
TYPE	Text (8)	Designed element thickness variation
TYPE	Text (8)	Element property type
PID	Integer > 0	Element property identification number

Created By: Module IFP

Entity: **DDWGH2**

Entity Type: Unstructured

Description: Contains the nonlinear portion of DWGH, the sensitivities of the weight to the design variables.

Entity Structure:

Record:

1. Design variable identification numbers including zeroth design variable.
2. The nonlinear portion of DWGH corresponding to each design variable ID at the 1st record.

Created By: Module NLEMA1

Notes:

The length of each record is NDV+1.

Entity: **DWGH1**

Entity Type: Unstructured

Description: Contains the linear design invariant portion of DWGH, the sensitivities of the weight to the design variables.

Entity Structure:

Record:

1. Design variable identification numbers including zeroth design variable.
2. The invariant portion of DWGH corresponding to each design variable ID at the 1st record.

Created By: Module EMA1

Notes:

The length of each record is NDV+1.

Entity: **DWNWSH**

Entity Type: Matrix

Description: Matrix containing downwash vectors that are computed for unit values of angle of attack, pitch rate and trim surface deflection.

Matrix Form: Rectangular real matrix with three columns and rows equal to the number of panels in the unsteady aerodynamics model.

Created By: Module BLASTFIT

Entity: **DYNRED**

Entity Type: Relation

Description: Contains the necessary information to perform general dynamic reduction as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
FMAX	Real > 0	Highest frequency of interest
NVEC	Integer > 0	Number of generalized coordinates desired
NIT	Integer > 0	Not Used
ISEED	Integer > 0	
NQDES	Integer > 0	
EPZ	Real	
FACTOR	Real	

Created By: Module IFP

Entity:	**D1JK**
Entity Type:	Matrix
Description:	The real part of the substantial derivative matrix.
Matrix Form:	A rectangular complex matrix with the number of rows equal to the number of aerodynamic panels and the number of columns equal to the number of aerodynamic degrees of freedom .
Created By:	Module UNSTEADY
Notes:	

1. The complete substantial derivative matrix is equal to:

   [D1JK] + (ik) **[D2JK]**

   where k is the reduced frequency.
2. The number of J degrees of freedom is INFO(11).
3. The number of K degrees of freedom is INFO(12).

Entity: **D2JK**

Entity Type: Matrix

Description: The imaginary part of the substantial derivative matrix.

Matrix Form: A complex matrix with the number of rows equal to the number of aerodynamic panels and the number of columns equal to the number of aerodynamic degrees of freedom.

Created By: Module UNSTEADY

Notes:

1. The complete substantial derivative matrix is equal to:

   [D1JK] + (iK) **[D2JK]**

   where k is the reduced frequency.

Entity: **EIDTYPE**

Entity Type: Relation

Description: Contains the list of element identification numbers and corresponding element types.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0	Element identification number
ETYPE	Text (8)	Element type

Created By: Module IFP

Entity: **EIGC**

Entity Type: Relation

Description: Contains the necessary information to perform complex eigenvalue analysis as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0, key	Set identification number
METHOD	Text (8)	Method of complex eigenvalue extraction
NORM	Text (8)	Eigenvector normalization technique
GRID1	Integer ≥ 0	Grid or scalar point identification number
COMPNTS1	Integer ≥ 0	Component of GRID1
ORTHPARM	Real > 0.0	Mass orthogonality test parameter
PA	Real	The real part of complex point A
QA	Real	The imaginary part of complex point B
PB	Real	The real part of complex point A
QB	Real	The imaginary part of complex point B
WIDTH	Real > 0.0	Width of region in complex plane
ROOTEST	Integer > 0	Estimated number of roots in the range
ROOTDES	Integer ≥ 0	Desired number of roots

Created By: Module IFP

Entity: **EIGR**

Entity Type: Relation

Description: Contains the necessary information to perform real eigenvalue analysis as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0, key	Set identification number
METHOD	Text (8)	Method of eigenvalue extraction
MINFREQ	Real ≥ 0.0	Lower bound for frequency
MAXFREQ	Real ≥ 0.0	Upper bound for frequency
ROOTEST1	Integer > 0	Estimated number of roots in the range
ROOTDES1	Integer ≥ 0	Desired number of roots
ORTHPARM	Real > 0.0	Mass orthogonality test parameter
NORM	Text (8)	Eigenvector normalization technique
GRID1	Integer ≥ 0	Grid or scalar point identification number
COMPNTS1	Integer ≥ 0	Component of GRID1

Created By: Module IFP

Entity: **ELAS**

Entity Type: Matrix

Description: Intermediate matrix in the nuclear blast response calculation containing the modal participation factors for the initially trimmed aircraft.

Matrix Form: ELAS contains one column and the number of rows is equal to the number of elastic modes.

Created By: Module MAPOL

Entity: **ELASEST**

Entity Type: Relation

Description: Contains the element summary data for the ELAS1 and ELAS2 elements.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
SIL1, SIL2	Integer ≥ 0	Internal grid or scalar point identification number
COMPNT1	Integer ≥ 0	Component of SIL1 to which the element is attached
COMPNT2	Integer ≥ 0	Component of SIL2 to which the element is attached
K	Real	Stiffness value
GE	Real	Damping coefficient
STRSCOEF	Real	Stress coefficient
DESIGN	Integer ≥ 0	Design flag, nonzero if element is designed

Created By: Module MAKEST

Notes:

1. This relation is built from the CELAS1 and CELAS2 relations along with associated property and grid relations. It contains one tuple for each scalar spring element in the problem.

Entity: **ELEMDRVA**

Entity Type: Relation

Description: Contains the user function requested element stress and strain response sensitivity information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
MXCOL	Integer > 0	Matrix column number in [MXELMDVA] for the sensitivity values

Created By: Module MKAMAT

Entity: **ELEMLIST**

Entity Type: Relation

Description: Contains the list of elements for which element dependent outputs are requested as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer > 0	Design variable identification number
ETYPE	Text (8)	Element type. One of the following: BAR QDMEM1 ELAS QUAD4 IHEX1 ROD IHEX2 SHEAR IHEX3 TRIA3 TRMEM
EID	Integer > 0	Element identification number

Created By: Module IFP

Entity: **ELEMRESP**

Entity Type: Relation

Description: Contains the user function requested element stress and strain response values.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
VALUE	Real	Response value

Created By: Module SCEVAL

Entity: **ELIST**

Entity Type: Relation

Description: Contains the element identification numbers of elements specified on the ELIST Bulk Data entry.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
LINKID	Integer > 0	ELIST set identification number
ETYPE1	Text (8)	Element type
EID1	Integer > 0	Element identification number

Created By: Module IFP

Notes:

1. Allowable ETYPE1 entries are:
   - CROD, CONROD
   - CSHEAR
   - CQDMEM1
   - CQUAD4
   - CTRIA3
   - CTRMEM
   - CMASS1, CMASS2
   - CBAR
   - CONM2
   - CELAS1, CELAS2

Entity: **ELISTM**

Entity Type: Relation

Description: Contains the element identification numbers of elements specified on the ELISTM Bulk Data entry.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
LINKID	Integer > 0	ELIST set identification number
ETYPE1	Text (8)	Element type
EID1	Integer > 0	Element identification number
DVSYMBL	Text(8)	Designed dimension symbol

Created By: Module IFP

Notes:

1. Allowable ETYPE1 entries are:
   - CROD, CONROD
   - CSHEAR
   - CQDMEM1
   - CQUAD4
   - CTRIA3
   - CTRMEM
   - CMASS1, CMASS2
   - CBAR
   - CONM2
   - CELAS1, CELAS2
2. Allowable DVSYMBL are:
   - D1, D2, D3, D4, D5, D6, D7, D8, D9, D10 for BAR element
   - cross section dimensions;
   - A for element cross section area;
   - T for element thickness;
   - M for element mass;
   - K for element thickness.

Entity: **EOBAR**

Entity Type: Relation

Description: Contains the element response quantities for the BAR element.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
OAFLAG	2 ≥ Integer ≥ 1	Optimize/analyze flag =1 Optimization =2 Analysis
NITER	Integer > 0	Iteration number for optimization
BCID	Integer > 0	Boundary condition identification number
DISC	Integer > 0	Discipline type = 1 Statics = 5 Transient = 2 Modes = 7 Buckling = 3 Steady Aero = 8 Blast
SUBCASE	Integer > 0	Subcase identification number if (DISC = 1, 3, 5, 8) or Mode Number if (DISC = 2, 7)
EID	Integer > 0	Element identification number
ETYPE	String	Element type ("BAR")
CMPLX	Integer > 0	Complex output identifier = 1 if real response quantities = 2 if complex response quantities
ESER	Real	Real part of element strain energy
ESEI	Real	Imaginary part of element strain energy
RSA1	Real	Real part of first bending stress at end A
ISA1	Real	Imaginary part of first bending stress at end A
RSA2	Real	Real part of second bending stress at end A
ISA2	Real	Imaginary part of second bending stress at end A
RSA3	Real	Real part of third bending stress at end A
ISA3	Real	Imaginary part of third bending stress at end A
RSA4	Real	Real part of fourth bending stress at end A
ISA4	Real	Imaginary part of fourth bending stress at end A
RAAX	Real	Real part of axial stress at end A
IAAX	Real	Imaginary part of axial stress at end A
MAXA	Real	Maximum stress at end A
MINA	Real	Minimum stress at end A

NAME	TYPE/KEY	DESCRIPTION
TSAFE	Real	Safety margin in tension
RSB1	Real	Real part of first bending stress at end B
ISB1	Real	Imaginary part of first bending stress at end B
RSB2	Real	Real part of second bending stress at end B
ISB2	Real	Imaginary part of second bending stress at end B
RSB3	Real	Real part of third bending stress at end B
ISB3	Real	Imaginary part of third bending stress at end B
RSB4	Real	Real part of fourth bending stress at end B
ISB4	Real	Imaginary part of fourth bending stress at end B
RBAX	Real	Real part of axial stress at end B
IBAX	Real	Imaginary part of axial stress at end B
MAXB	Real	Maximum stress at end B
MINB	Real	Minimum stress at end B
CSAFE	Real	Safety margin in compression
RSNA1	Real	Real part of first bending strain at end A
ISNA1	Real	Imaginary part of first bending strain at end A
RSNA2	Real	Real part of second bending strain at end A
ISNA2	Real	Imaginary part of second bending strain at end A
RSNA3	Real	Real part of third bending strain at end A
ISNA3	Real	Imaginary part of third bending strain at end A
RSNA4	Real	Real part of fourth bending strain at end A
ISNA4	Real	Imaginary part of fourth bending strain at end A
RAAXN	Real	Real part of axial strain at end A
IAAXN	Real	Imaginary part of axial strain at end A
MAXAN	Real	Maximum strain at end A
MINAN	Real	Minimum strain at end A
RSNB1	Real	Real part of first bending strain at end B
ISNB1	Real	Imaginary part of first bending strain at end B
RSNB2	Real	Real part of second bending strain at end B

NAME	TYPE/KEY	DESCRIPTION
ISNB2	Real	Imaginary part of second bending strain at end B
RSNB3	Real	Real part of third bending strain at end B
ISNB3	Real	Imaginary part of third bending strain at end B
RSNB4	Real	Real part of fourth bending strain at end B
ISNB4	Real	Imaginary part of fourth bending strain at end B
RBAXN	Real	Real part of axial strain at end B
IBAXN	Real	Imaginary part of axial strain at end B
MAXBN	Real	Maximum strain at end B
MINBN	Real	Minumum strain at end B
RBMA1	Real	Real part of bending moment A1
IBMA1	Real	Imaginary part of bending moment A1
RBMA2	Real	Real part of bending moment A2
IBMA2	Real	Imaginary part of bending moment A2
RBMB1	Real	Real part of bending moment B1
IBMB1	Real	Imaginary part of bending moment B1
RBMB2	Real	Real part of bending moment B2
IBMB2	Real	Imaginary part of bending moment B2
RSHEAR1	Real	Real part of shear 1
ISHEAR1	Real	Imaginary part of shear 1
RSHEAR2	Real	Real part of shear 2
ISHEAR2	Real	Imaginary part of shear 2
RFORAX	Real	Real part of axial force
IFORAX	Real	Imaginary part of axial force
RTORQUE	Real	Real part of torque
ITORQUE	Real	Imaginary part of torque

Created By: Module EDR

Notes:

1. This relation is used by module OFPEDR for output printing and punching.

Entity: **EODISC**

Entity Type: Unstructured

Description: Contains the element discipline types and their subcases for which element response quantities are to be computed for each element in the structural model for each boundary condition.

Record:

i. Record i contains the following for each EID/BCID combination to the EOSUMMARY relation.

WORD	CONTENTS
1	NDISC, the number of disciplines in the EODISC record
2	DISC, discipline ID for the current discipline
3	NSUB, the number of subcases for which output is desired from discipline DISC
4 to 3+NSUB	SUB_i, the subcase numbers in sorted order

Created By: Module PFBULK

Notes:

1. Words 2 through 4+NSUB are repeated for each of the NDISC disciplines to generate a record in the form.

   $NDISC\ (DISC_i,\ NSUB_i,\ (SUB_j),\ j=1,NSUB),\ i=1,\ NDISC)$

2. Each record of EODISC is referenced by the RECORD attribute of the EOSUMMARY relation.
3. The EOSUMMARY/EODISC combination is used by EDR and OFPEDR to control element response quantity computations.
4. Each record is ordered in discipline, in subcase order.
5. The records are ordered by boundary condition ID, element type (alphabetical) and element ID.

Entity: **EOELAS**

Entity Type: Relation

Description: Contains the element response quantities for the ELAS element.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
OAFLAG	2 ≥ Integer ≥ 1	Optimize/analyze flag =1 Optimization =2 Analysis
NITER	Integer > 0	Iteration number for optimization
BCID	Integer > 0	Boundary condition identification number
DISC	Integer > 0	Discipline type = 1 Statics = 5 Transient = 2 Modes = 7 Buckling = 3 Steady Aero = 8 Blast
SUBCASE	Integer > 0	Subcase identification number if (DISC = 1, 3, 5, 8) or Mode Number if (DISC = 2, 7)
EID	Integer > 0	Element identification number
ETYPE	String	Element type ("ELAS")
CMPLX	Integer > 0	Complex output identifier = 1 if real response quantities = 2 if complex response quantities
ESER	Real	Real part of element strain energy
ESEI	Real	Imaginary part of element strain energy
STRSR	Real	Real part of stress
STRSI	Real	Imaginary part of stress
FORR	Real	Real part of force
FORI	Real	Imaginary part of force

Created By: Module EDR

Notes:

1. This relation is used by module OFPEDR for output printing and punching.

Entity: **EOHEX1**

Entity Type: Relation

Description: Contains the element response quantities for the IHEX1 element.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
OAFLAG	2 ≥ Integer ≥ 1	Optimize/analyze flag =1 Optimization =2 Analysis
NITER	Integer > 0	Iteration number for optimization
BCID	Integer > 0	Boundary condition identification number
DISC	Integer > 0	Discipline type = 1 Statics = 5 Transient = 2 Modes = 7 Buckling = 3 Steady Aero = 8 Blast
SUBCASE	Integer > 0	Subcase identification number if (DISC = 1, 3, 5, 8) or Mode Number if (DISC = 2, 7)
EID	Integer > 0	Element identification number
ETYPE	String	Element type ("IHEX1")
CMPLX	Integer > 0	Complex output identifier = 1 if real response quantities = 2 if complex response quantities
GID	Integer > 0	Stress point identification number
ESER	Real	Real part of element strain energy
ESEI	Real	Imaginary part of element strain energy
RSTRSX	Real	Real part of normal stress in x-direction
ISTRSX	Real	Imaginary part of normal stress in x-direction
RSSXY	Real	Real part of shear stress in xy-plane
ISSXY	Real	Imaginary part of shear stress in xy-plane
PSTRESS1	Real	First principal stress
XCOS1	Real	First principal x cosine
XCOS2	Real	Second principal x cosine
XCOS3	Real	Third principal x cosine
MEANSTRS	Real	Mean stress
OCTSTRS	Real	Octahedral shear stress
RSTRSY	Real	Real part of normal stress in y-direction

NAME	TYPE/KEY	DESCRIPTION
ISTRSY	Real	Imaginary part of normal stress in y-direction
RSSYZ	Real	Real part of normal stress in yz-direction
ISSYZ	Real	Imaginary part of normal stress in yz-direction
PSTRESS2	Real	Second principal stress
YCOS1	Real	First principal y cosine
YCOS2	Real	Second principal y cosine
YCOS3	Real	Third principal y cosine
RSTRSZ	Real	Real part of normal stress in z-direction
ISTRSZ	Real	Imaginary part of normal stress in z-direction
RSSZX	Real	Real part of shear stress in zx-plane
ISSZX	Real	Imaginary part of shear stress in zx-plane
PSTRESS3	Real	Third principal stress
ZCOS1	Real	First principal z cosine
ZCOS2	Real	Second principal z cosine
ZCOS3	Real	Third principal z cosine
RSTRNX	Real	Real part of normal strain in x-direction
ISTRNX	Real	Imaginary part of normal strain in x-direction
RSNXY	Real	Real part of shear strain in xy-plane
ISNXY	Real	Imaginary part of shear strain in xy-plane
PSTRAIN1	Real	First principal strain
XCOS1N	Real	First principal x cosine
XCOS2N	Real	Second principal x cosine
XCOS3N	Real	Third principal x cosine
MEANSTRN	Real	Mean strain
OCTSTRN	Real	Octahedral shear strain
RSTRNY	Real	Real part of normal strain in y-direction
ISTRNY	Real	Imaginary part of normal strain in y-direction
RSNYZ	Real	Real part of shear strain in yz-plane
ISNYZ	Real	Imaginary part of shear strain in yz-plane

NAME	TYPE/KEY	DESCRIPTION
PSTRAIN2	Real	Second principal strain
YCOS1N	Real	First principal y cosine
YCOS2N	Real	Second principal y cosine
YCOS3N	Real	Third principal y cosine
RSTRNZ	Real	Real part of normal strain in z-direction
ISTRNZ	Real	Imaginary part of normal strain in z-direction
RSNZX	Real	Real part of shear strain in zx-plane
ISNZX	Real	Imaginary part of shear strain in zx-plane
PSTRAIN3	Real	Third principal strain
ZCOS1N	Real	First principal z cosine
ZCOS2N	Real	Second principal z cosine
ZCOS3N	Real	Third principal z cosine

Created By: Module EDR

Notes:

1. This relation is used by module OFPEDR for output printing and punching.
2. One tuple exists for each of the nine stress points in the element.
3. The first eight stress points (attribute GID) are coincident with the element grid points and are numbered 1 through 8 in the order that the grid points are specified on the CIHEX1 entity. The ninth stress point (GID=9) is located at the center of the element.

Entity: **EOHEX2**

Entity Type: Relation

Description: Contains the element response quantities for the IHEX2 element.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
OAFLAG	2 ≥ Integer ≥ 1	Optimize/analyze flag =1 Optimization =2 Analysis
NITER	Integer > 0	Iteration number for optimization
BCID	Integer > 0	Boundary condition identification number
DISC	Integer > 0	Discipline type = 1 Statics = 5 Transient = 2 Modes = 7 Buckling = 3 Steady Aero = 8 Blast
SUBCASE	Integer > 0	Subcase identification number if (DISC = 1, 3, 5, 8) or Mode Number if (DISC = 2, 7)
EID	Integer > 0	Element identification number
ETYPE	String	Element type ("IHEX2")
CMPLX	Integer > 0	Complex output identifier = 1 if real response quantities = 2 if complex response quantities
GID	Integer > 0	Stress point identification number
ESER	Real	Real part of element strain energy
ESEI	Real	Imaginary part of element strain energy
RSTRSX	Real	Real part of normal stress in x-direction
ISTRSX	Real	Imaginary part of normal stress in x-direction
RSSXY	Real	Real part of shear stress in xy-plane
ISSXY	Real	Imaginary part of shear stress in xy-plane
PSTRESS1	Real	First principal stress
XCOS1	Real	First principal x cosine
XCOS2	Real	Second principal x cosine
XCOS3	Real	Third principal x cosine
MEANSTRS	Real	Mean stress
OCTSTRS	Real	Octahedral shear stress
RSTRSY	Real	Real part of normal stress in y-direction

NAME	TYPE/KEY	DESCRIPTION
ISTRSY	Real	Imaginary part of normal stress in y-direction
RSSYZ	Real	Real part of normal stress in yz-plane
ISSYZ	Real	Imaginary part of normal stress in yz-plane
PSTRESS2	Real	Second principal stress
YCOS1	Real	First principal y cosine
YCOS2	Real	Second principal y cosine
YCOS3	Real	Third principal y cosine
RSTRSZ	Real	Real part of normal stress in z-direction
ISTRSZ	Real	Imaginary part of normal stress in z-direction
RSSZX	Real	Real part of shear stress in xy-plane
ISSZX	Real	Imaginary part of shear stress in xy-plane
PSTRESS3	Real	Third principal stress
ZCOS1	Real	First principal z cosine
ZCOS2	Real	Second principal z cosine
ZCOS3	Real	Third principal z cosine
RSTRNX	Real	Real part of normal strain in x-direction
ISTRNX	Real	Imaginary part of normal strain in x-direction
RSNXY	Real	Real part of shear strain in xy-plane
ISNXY	Real	Imaginary part of shear strain in xy-plane
PSTRAIN1	Real	First principal strain
XCOS1N	Real	First principal x cosine
XCOS2N	Real	Second principal x cosine
XCOS3N	Real	Third principal x cosine
MEANSTRN	Real	Mean strain
OCTSTRN	Real	Octahedral shear strain
RSTRNY	Real	Real part of normal strain in y-direction
ISTRNY	Real	Imaginary part of normal strain in y-direction
RSNYZ	Real	Real part of shear strain in yz-plane
ISNYZ	Real	Imaginary part of shear strain in yz-plane
PSTRAIN2	Real	Second principal strain

NAME	TYPE/KEY	DESCRIPTION
YCOS1N	Real	First principal y cosine
YCOS2N	Real	Second principal y cosine
YCOS3N	Real	Third principal y cosine
RSTRNZ	Real	Real part of normal strain in z-direction
ISTRNZ	Real	Imaginary part of normal strain in z-direction
RSTZX	Real	Real part of shear strain in z-direction
ISNZX	Real	Imaginary part of shear strain in z-direction
PSTRAIN3	Real	Third principal strain
ZCOS1N	Real	First principal z cosine
ZCOS2N	Real	Second principal z cosine
ZCOS3N	Real	Third principal z cosine

Created By: Module EDR

Notes:

1. This relation is used by module OFPEDR for output printing and punching.
2. One tuple exists for each of the nine stress points in the element.
3. The first 20 stress points are in the same order as the grid points are specified on the CIHEX2 entity and are numbered 1 through 20. The corner stress points coincident with the corner grid points while the mid-edge stress points are exactly at the mid-edge point. The 21st stress point is located at the element center.

Entity: **EOHEX3**

Entity Type: Relation

Description: Contains the element response quantities for the IHEX3 element.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
OAFLAG	2 ≥ Integer ≥ 1	Optimize/analyze flag =1 Optimization =2 Analysis
NITER	Integer > 0	Iteration number for optimization
BCID	Integer > 0	Boundary condition identification number
DISC	Integer > 0	Discipline type = 1 Statics = 5 Transient = 2 Modes = 7 Buckling = 3 Steady Aero = 8 Blast
SUBCASE	Integer > 0	Subcase identification number if (DISC = 1, 3, 5, 8) or Mode Number if (DISC = 2, 7)
EID	Integer > 0	Element identification number
ETYPE	String	Element type ("IHEX3")
CMPLX	Integer > 0	Complex output identifier = 1 if real response quantities = 2 if complex response quantities
GID	Integer > 0	Stress point identification number
ESER	Real	Real part of element strain energy
ESEI	Real	Imaginary part of element strain energy
RSTRSX	Real	Real part of normal stress in x-direction
ISTRSX	Real	Imaginary part of normal stress in x-direction
RSSXY	Real	Real part of shear stress in xy-plane
ISSXY	Real	Imaginary part of shear stress in xy-plane
PSTRESS1	Real	First principal stress
XCOS1	Real	First principal x cosine
XCOS2	Real	Second principal x cosine
XCOS3	Real	Third principal x cosine
MEANSTRS	Real	Mean stress
OCTSTRS	Real	Octahedral shear stress
RSTRSY	Real	Real part of normal stress in y-direction

NAME	TYPE/KEY	DESCRIPTION
ISTRSY	Real	Imaginary part of normal stress in y-direction
RSSYZ	Real	Real part of normal stress in yz-plane
ISSYZ	Real	Imaginary part of normal stress in yz-plane
PSTRESS2	Real	Second principal stress
YCOS1	Real	First principal y cosine
YCOS2	Real	Second principal y cosine
YCOS3	Real	Third principal y cosine
RSTRSZ	Real	Real part of normal stress in z-direction
ISTRSZ	Real	Imaginary part of normal stress in z-direction
RSSZX	Real	Real part of shear stress in xy-plane
ISSZX	Real	Imaginary part of shear stress in xy-plane
PSTRESS3	Real	Third principal stress
ZCOS1	Real	First principal z cosine
ZCOS2	Real	Second principal z cosine
ZCOS3	Real	Third principal z cosine
RSTRNX	Real	Real part of normal strain in x-direction
ISTRNX	Real	Imaginary part of normal strain in x-direction
RSNXY	Real	Real part of shear strain in xy-plane
ISNXY	Real	Imaginary part of shear strain in xy-plane
PSTRAIN1	Real	First principal strain
XCOS1N	Real	First principal x cosine
XCOS2N	Real	Second principal x cosine
XCOS3N	Real	Third principal x cosine
MEANSTRN	Real	Mean strain
OCTSTRN	Real	Octahedral shear strain
RSTRNY	Real	Real part of normal strain in y-direction
ISTRNY	Real	Imaginary part of normal strain in y-direction
RSNYZ	Real	Real part of shear strain in yz-plane
ISNYZ	Real	Imaginary part of shear strain in yz-plane
PSTRAIN2	Real	Second principal strain

NAME	TYPE/KEY	DESCRIPTION
YCOS1N	Real	First principal y cosine
YCOS2N	Real	Second principal y cosine
YCOS3N	Real	Third principal y cosine
RSTRNZ	Real	Real part of normal strain in z-direction
ISTRNZ	Real	Imaginary part of normal strain in z-direction
RSTZX	Real	Real part of shear strain in z-direction
ISNZX	Real	Imaginary part of shear strain in z-direction
PSTRAIN3	Real	Third principal strain
ZCOS1N	Real	First principal z cosine
ZCOS2N	Real	Second principal z cosine
ZCOS3N	Real	Third principal z cosine

Created By: Module EDR

Notes:

1. This relation is used by module OFPEDR for output printing and punching.
2. One tuple exists for each of the nine stress points in the element.
3. The first 20 stress points are in the same order as the grid points are specified on the CIHEX3 entity and are numbered 1 through 20. The corner stress points coincident with the corner grid points while the mid-edge stress points are exactly at the mid-edge point. The 21st stress point is located at the element center.

Entity: **EOQDMM1**

Entity Type: Relation

Description: Contains the element response quantities for the

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
OAFLAG	2 ≥ Integer ≥ 1	Optimize/analyze flag =1 Optimization =2 Analysis
NITER	Integer > 0	Iteration number for optimization
BCID	Integer > 0	Boundary condition identification number
DISC	Integer > 0	Discipline type = 1 Statics = 5 Transient = 2 Modes = 7 Buckling = 3 Steady Aero = 8 Blast
SUBCASE	Integer > 0	Subcase identification number if (DISC = 1, 3, 5, 8) or Mode Number if (DISC = 2, 7)
EID	Integer > 0	Element identification number
ETYPE	String	Element type ("QDMEM1")
CMPLX	Integer > 0	Complex output identifier = 1 if real response quantities = 2 if complex response quantities
LAYRNUM	Integer > 0	Layer number
ESER	Real	Real part of element strain energy
ESEI	Real	Imaginary part of element strain energy
RSTRSX	Real	Real part of normal stress in x-direction
ISTRSX	Real	Imaginary part of normal stress in x-direction
RSTRSY	Real	Real part of normal stress in y-direction
ISTRSY	Real	Imaginary part of normal stress in y-direction
RSTRSS	Real	Real part of shear stress
ISTRSS	Real	Imaginary part of shear stress
THSTRS	Real	Principal angle for stress
STRS1	Real	Major principal stress
STRS2	Real	Minor principal stress
MSSTRS	Real	Maximum shear stress
RSTRNX	Real	Real part of normal strain in x-direction

NAME	TYPE/KEY	DESCRIPTION
ISTRNX	Real	Imaginary part of normal strain in x-direction
RSTRNY	Real	Real part of normal strain in y-direction
ISTRNY	Real	Imaginary part of normal strain in y-direction
RSTRNS	Real	Real part of shear strain
ISTRNS	Real	Imaginary part of shear strain
THSTRN	Real	Principal angle for strain
STRN1	Real	Major principal strain
STRN2	Real	Minor principal atrain
MSSTRN	Real	Maximum shear strain
RFX	Real	Real part of force in x-direction
IFX	Real	Imaginary part of force in x-direction
RFY	Real	Real part of force in y-direction
IFY	Real	Imaginary part of force in y-direction
RFXY	Real	Real part of shear force in xy-plane
IFXY	Real	Imaginary part of shear force in xy-plane

Created By: Module EDR

Notes:

1. This relation is used by module OFPEDR for output printing and punching.

Entity: **EOQUAD4**

Entity Type: Relation

Description: Contains the element response quantities for the QUAD4 element.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
OAFLAG	2 ≥ Integer ≥ 1	Optimize/analyze flag =1 Optimization =2 Analysis
NITER	Integer > 0	Iteration number for optimization
BCID	Integer > 0	Boundary condition identification number
DISC	Integer > 0	Discipline type = 1 Statics = 5 Transient = 2 Modes = 7 Buckling = 3 Steady Aero = 8 Blast
SUBCASE	Integer > 0	Subcase identification number if (DISC = 1, 3, 5, 8) or Mode Number if (DISC = 2, 7)
EID	Integer > 0	Element identification number
ETYPE	String	Element type ("QUAD4")
CMPLX	Integer > 0	Complex output identifier = 1 if real response quantities = 2 if complex response quantities
LAYRNUM	Integer > 0	Layer number
CMPZIT	Integer ≥ 0	Composite type flag = 0 Noncomposite element = 1 Nondesigned and nonconstrained composite element = 2 Nondesigned and constrained composite element = 3 Designed membrane composite element = 4 Designed bending composite element
ESER	Real	Real part of element strain energy
ESEI	Real	Imaginary part of element strain energy
Z1	Real	Fiber distance 1
RSTRSX1	Real	Real part of normal stress in x-direction at Z1
ISTRSX1	Real	Imaginary part of normal stress in x-direction at Z1
RSTRSY1	Real	Real part of stress in y-direction at Z1

NAME	TYPE/KEY	DESCRIPTION
ISTRSY1	Real	Imaginary part of stress in y-direction at Z1
RSSXY1	Real	Real part of shear stress at Z1
ISSXY1	Real	Imaginary part of shear stress at Z1
ANGLES1	Real	Principal angle for stress at Z1
STRS11	Real	Major principal stress at Z1
STRS12	Real	Minor principal stress at Z1
MAXSS1	Real	Maximum shear stress at Z1
Z2	Real	Fiber distance 2
RSTRSX2	Real	Real part of stress in x-direction at Z2
ISTRSX2	Real	Imaginary part of stress in x-direction at Z2
RSTRSY2	Real	Real part of stress in y-direction at Z2
ISTRSY2	Real	Imaginary part of stress in y-direction at Z2
RSSXY2	Real	Real part of shear stress at Z2
ISSXY2	Real	Imaginary part of shear stress at Z2
ANGLES2	Real	Principal angle for stress at Z2
STRS21	Real	Major principal stress at Z2
STRS22	Real	Minor principal stress at Z2
MAXSS2	Real	Maximum shear stress at Z2
RSTRNX1	Real	Real part of strain in x-direction at Z1
ISTRNX1	Real	Imaginary part of strain in x-direction at Z1
RSTRNY1	Real	Real part of strain in y-direction at Z1
ISTRNY1	Real	Imaginary part of strain in y-direction at Z1
RSNXY1	Real	Real part of shear strain at Z1
ISNXY1	Real	Imaginary part of shear strain at Z1
ANGLEN1	Real	Principal axis angle at Z1
STRN11	Real	Major principal strain at Z1
STRN12	Real	Minor principal strain at Z1
MAXNS1	Real	Maximum shear strain at Z1
RSTRNX2	Real	Real part of strain in x-direction at Z2
ISTRNX2	Real	Imaginary part of strain in x-direction at Z2
RSTRNY2	Real	Real part of strain in y-direction at Z2
ISTRNY2	Real	Imaginary part of strain in y-direction at Z2

NAME	TYPE/KEY	DESCRIPTION
RSNXY2	Real	Real part of shear strain at Z2
ISNXY2	Real	Imaginary part of shear strain at Z2
ANGELN2	Real	Principal axis angle at Z2
STRN21	Real	Major principal strain at Z2
STRN22	Real	Minor principal strain at Z2
MAXSN2	Real	Maximum shear strain at Z2
RMEMX	Real	Real part of membrane force in x-direction
IMEMX	Real	Imaginary part of membrane force in x-direction
RMEMY	Real	Real part of membrane force in y-direction
IMEMY	Real	Imaginary part of membrane force in y-direction
RMEMXY	Real	Real part of membrane force in xy-plane
IMEMXY	Real	Imaginary part of membrane force in xy-plane
RBENDX	Real	Real part of bending moment in x-direction
IBENDX	Real	Imaginary part of bending moment in x-direction
RBENDY	Real	Real part of bending moment in y-direction
IBENDY	Real	Imaginary part of bending moment in y-direction
RBENDXY	Real	Real part of bending moment in xy-plane
IBENDXY	Real	Imaginary part of bending moment in xy-plane
RSHEARX	Real	Real part of shear force in x-direction
ISHEARX	Real	Imaginary part of shear force in x-direction
RSHEARY	Real	Real part of shear force in y-direction
ISHEARY	Real	Imaginary part of shear force in y-direction

Created By: Module EDR

Notes:

1. This relation is used by module OFPEDR for printing and punching.

Entity: **EOROD**

Entity Type: Relation

Description: Contains the element response quantities for the ROD element.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
OAFLAG	2 ≥ Integer ≥ 1	Optimize/analyze flag =1 Optimization =2 Analysis
NITER	Integer > 0	Iteration number for optimization
BCID	Integer > 0	Boundary condition identification number
DISC	Integer > 0	Discipline type = 1 Statics = 5 Transient = 2 Modes = 7 Buckling = 3 Steady Aero = 8 Blast
SUBCASE	Integer > 0	Subcase identification number if (DISC = 1, 3, 5, 8) or Mode Number if (DISC = 2, 7)
EID	Integer > 0	Element identification number
ETYPE	String	Element type ("ROD")
CMPLX	Integer > 0	Complex output identifier = 1 if real response quantities = 2 if complex response quantities
ESER	Real	Real part of element strain energy
ESEI	Real	Imaginary part of element strain energy
RAXSTRS	Real	Real part of axial stress
IAXSTRS	Real	Imaginary part of axial stress
RTORSTRS	Real	Real part of torsional stress
ITORSTRS	Real	Imaginary part of torsional stress
AMARSTRS	Real	Axial stress margin of safety
TMARSTRS	Real	Torsional stress margin of safety
RAXSTRN	Real	Real part of axial strain
IAXSTRN	Real	Imaginary part of axial strain
RTORSTRN	Real	Real part of torsional strain
ITORSTRN	Real	Imaginary part of torsional strain
RAXFOR	Real	Real part of axial force
IAXFOR	Real	Imaginary part of axial force
RTORQUE	Real	Real part of torsional force
ITORQUE	Real	Imaginary part of torsional force

Created By: Module EDR

Notes:

1. This relation is used by module OFPEDR for printing and punching.

Entity: **EOSHEAR**

Entity Type: Relation

Description: Contains the element response quantities for the SHEAR element.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
OAFLAG	2 ≥ Integer ≥ 1	Optimize/analyze flag =1 Optimization =2 Analysis
NITER	Integer > 0	Iteration number for optimization
BCID	Integer > 0	Boundary condition identification number
DISC	Integer > 0	Discipline type = 1 Statics = 5 Transient = 2 Modes = 7 Buckling = 3 Steady Aero = 8 Blast
SUBCASE	Integer > 0	Subcase identification number if (DISC = 1, 3, 5, 8) or Mode Number if (DISC = 2, 7)
EID	Integer > 0	Element identification number
ETYPE	String	Element type ("SHEAR")
CMPLX	Integer > 0	Complex output identifier = 1 if real response quantities = 2 if complex response quantities
ESER	Real	Real part of element strain energy
ESEI	Real	Imaginary part of element strain energy
RMAXSTRS	Real	Real part of maximum shear stress
IMAXSTRS	Real	Imaginary part of maximum shear stress
RAVESTRS	Real	Real part of average shear stress
IAVESTRS	Real	Imaginary part of average shear stress
SMARSTRS	Real	Stress safety margin
RMAXSTRN	Real	Real part of maximum shear strain
IMAXSTRN	Real	Imaginary part of maximum shear strain
RAVESTRN	Real	Real part of average shear strain
IAVESTRN	Real	Imaginary part of average shear strain
RF14	Real	Real part of normal force on 1 from 4
IF14	Real	Imaginary part of normal force on 1 from 4
RF12	Real	Real part of normal force on 1 from 2
IF12	Real	Imaginary part of normal force on 1 from 2
RF21	Real	Real part of normal force on 2 from 1

NAME	TYPE/KEY	DESCRIPTION
IF21	Real	Imaginary part of normal force on 2 from 1
RF23	Real	Real part of normal force on 2 from 3
IF23	Real	Imaginary part of normal force on 2 from 3
RF32	Real	Real part of normal force on 3 from 2
IF32	Real	Imaginary part of normal force on 3 from 2
RF34	Real	Real part of normal force on 3 from 4
IF34	Real	Imaginary part of normal force on 3 from 4
RF43	Real	Real part of normal force on 4 from 3
IF43	Real	Imaginary part of normal force on 4 from 3
RF41	Real	Real part of normal force on 4 from 1
IF41	Real	Imaginary part of normal force on 4 from 1
RK1	Real	Real part of shear panel kick force at 1
IK1	Real	Imaginary part of shear panel kick force at 1
RS12	Real	Real part of shear force 1-2
IS12	Real	Imaginary part of shear force 1-2
RK2	Real	Real part of shear panel kick force at 2
IK2	Real	Imaginary part of shear panel kick force at 2
RS23	Real	Real part of shear force 2-3
IS23	Real	Imaginary part of shear force 2-3
RK3	Real	Real part of shear panel kick force at 3
IK3	Real	Imaginary part of shear panel kick force at 3
RS34	Real	Real part of shear force 3-4
IS34	Real	Imaginary part of shear force 3-4
RK4	Real	Real part of shear panel kick force at 4
IK4	Real	Imaginary part of shear panel kick force at 4
RS41	Real	Real part of shear force 4-1
IS41	Real	Imaginary part of shear force 4-1

Created By: Module EDR

Notes:

1. This relation is used by module OFPEDR for output printing and punching.

Entity: **EOSUMMRY**

Entity Type: Relation

Description: Contains a summary of the element output requests.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
BCID	Integer > 0	Boundary condition identification number
NITER	Integer ≥ –1	Iteration step for output (–1 for all)
EID	Integer > 0	Element identification number
ETYPE	Text (8)	Element type (example: "ROD")
RECORD	Integer > 0	Record number in EODISC unstructured entity containing related data

Created By: Module PFBULK

Notes:

1. For each BCID, the tuples are sorted by ETYPE and then EID.

Entity: **EOTRIA3**

Entity Type: Relation

Description: Contains the element response quantities for the TRIA3 element.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
OAFLAG	2 ≥ Integer ≥ 1	Optimize/analyze flag =1 Optimization =2 Analysis
NITER	Integer > 0	Iteration number for optimization
BCID	Integer > 0	Boundary condition identification number
DISC	Integer > 0	Discipline type = 1 Statics = 5 Transient = 2 Modes = 7 Buckling = 3 Steady Aero = 8 Blast
SUBCASE	Integer > 0	Subcase identification number if (DISC = 1, 3, 5, 8) or Mode Number if (DISC = 2, 7)
EID	Integer > 0	Element identification number
ETYPE	String	Element type ("TRIA3")
CMPLX	Integer > 0	Complex output identifier = 1 if real response quantities = 2 if complex response quantities
LAYRNUM	Integer > 0	Layer number
CMPZIT	Integer ≥ 0	Composite type flag = 0 Noncomposite element = 1 Nondesigned and nonconstrained composite element = 2 Nondesigned and constrained composite element = 3 Designed membrane composite element = 4 Designed bending composite element
ESER	Real	Real part of element strain energy
ESEI	Real	Imaginary part of element strain energy
Z1	Real	Fiber distance 1
RSTRSX1	Real	Real part of normal stress in x-direction at Z1
ISTRSX1	Real	Imaginary part of normal stress in x-direction at Z1
RSTRSY1	Real	Real part of stress in y-direction at Z1

NAME	TYPE/KEY	DESCRIPTION
ISTRSY1	Real	Imaginary part of stress in y-direction at Z1
RSSXY1	Real	Real part of shear stress at Z1
ISSXY1	Real	Imaginary part of shear stress at Z1
ANGLES1	Real	Principal angle for stress at Z1
STRS11	Real	Major principal stress at Z1
STRS12	Real	Minor principal stress at Z1
MAXSS1	Real	Maximum shear stress at Z1
Z2	Real	Fiber distance 2
RSTRSX2	Real	Real part of stress in x-direction at Z2
ISTRSX2	Real	Imaginary part of stress in x-direction at Z2
RSTRSY2	Real	Real part of stress in y-direction at Z2
ISTRSY2	Real	Imaginary part of stress in y-direction at Z2
RSSXY2	Real	Real part of shear stress at Z2
ISSXY2	Real	Imaginary part of shear stress at Z2
ANGLES2	Real	Principal angle for stress at Z2
STRS21	Real	Major principal stress at Z2
STRS22	Real	Minor principal stress at Z2
MAXSS2	Real	Maximum shear stress at Z2
RSTRNX1	Real	Real part of strain in x-direction at Z1
ISTRNX1	Real	Imaginary part of strain in x-direction at Z1
RSTRNY1	Real	Real part of strain in y-direction at Z1
ISTRNY1	Real	Imaginary part of strain in y-direction at Z1
RSNXY1	Real	Real part of shear strain at Z1
ISNXY1	Real	Imaginaryl part of shear strain at Z1
ANGLEN1	Real	Principal axis angle at Z1
STRN11	Real	Major principal strain at Z1
STRN12	Real	Minor principal strain at Z1
MAXNS1	Real	Maximum shear strain at Z1
RSTRNX2	Real	Real part of strain in x-direction at Z2
ISTRNX2	Real	Imaginary part of strain in x-direction at Z2
RSTRNY2	Real	Real part of strain in y-direction at Z2
ISTRNY2	Real	Imaginary part of strain in y-direction at Z2

NAME	TYPE/KEY	DESCRIPTION
RSNXY2	Real	Real part of shear strain at Z2
ISNXY2	Real	Imaginary part of shear strain at Z2
ANGELN2	Real	Principal axis angle at Z2
STRN21	Real	Major principal strain at Z2
STRN22	Real	Minor principal strain at Z2
MAXSN2	Real	Maximum shear strain at Z2
RMEMX	Real	Real part of membrane force in x-direction
IMEMX	Real	Imaginary part of membrane force in x-direction
RMEMY	Real	Real part of membrane force in y-direction
IMEMY	Real	Imaginary part of membrane force in y-direction
RMEMXY	Real	Real part of membrane force in xy-plane
IMEMXY	Real	Imaginary part of membrane force in xy-plane
RBENDX	Real	Real part of bending moment in x-direction
IBENDX	Real	Imaginary part of bending moment in x-direction
RBENDY	Real	Real part of bending moment in y-direction
IBENDY	Real	Imaginary part of bending moment in y-direction
RBENDXY	Real	Real part of bending moment in xy-plane
IBENDXY	Real	Imaginary part of bending moment in xy-plane
RSHEARX	Real	Real part of shear force in x-direction
ISHEARX	Real	Imaginary part of shear force in x-direction
RSHEARY	Real	Real part of shear force in y-direction
ISHEARY	Real	Imaginary part of shear force in y-direction

Created By: Module EDR

Notes:

1. This relation is used by module OFPEDR for output printing and punching.

Entity: **EOTRMEM**

Entity Type: Relation

Description: Contains the element response quantities for the TRMEM element.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
OAFLAG	2 ≥ Integer ≥ 1	Optimize/analyze flag =1 Optimization =2 Analysis
NITER	Integer > 0	Iteration number for optimization
BCID	Integer > 0	Boundary condition identification number
DISC	Integer > 0	Discipline type = 1 Statics = 5 Transient = 2 Modes = 7 Buckling = 3 Steady Aero = 8 Blast
SUBCASE	Integer > 0	Subcase identification number if (DISC = 1, 3, 5, 8) or Mode Number if (DISC = 2, 7)
EID	Integer > 0	Element identification number
ETYPE	String	Element type ("TRMEM")
CMPLX	Integer > 0	Complex output identifier = 1 if real response quantities = 2 if complex response quantities
LAYRNUM	Integer > 0	Layer number
ESER	Real	Real part of element strain energy
ESEI	Real	Imaginary part of element strain energy
RSTRSX	Real	Real part of normal stress in x-direction
ISTRSX	Real	Imaginary part of normal stress in x-direction
RSTRSY	Real	Real part of normal stress in y-direction
ISTRSY	Real	Imaginary part of normal stress in y-direction
RSTRSS	Real	Real part of shear stress
ISTRSS	Real	Imaginary part of shear stress
THSTRS	Real	Principal angle for stress
STRS1	Real	Major principal stress
STRS2	Real	Minor principal stress
MSSTRS	Real	Maximum shear stress
RSTRNX	Real	Real part of normal strain in x-direction

NAME	TYPE/KEY	DESCRIPTION
ISTRNX	Real	Imaginary part of normal strain in x-direction
RSTRNY	Real	Real part of normal strain in y-direction
ISTRNY	Real	Imaginary part of normal strain in y-direction
RSTRNS	Real	Real part of shear strain
ISTRNS	Real	Imaginary part of shear strain
THSTRN	Real	Principal angle for strain
STRN1	Real	Major principal strain
STRN2	Real	Minor principal atrain
MSSTRN	Real	Maximum shear strain
RFX	Real	Real part of force in x-direction
IFX	Real	Imaginary part of force in x-direction
RFY	Real	Real part of force in y-direction
IFY	Real	Imaginary part of force in y-direction
RFXY	Real	Real part of shear force in xy-plane
IFXY	Real	Imaginary part of shear force in xy-plane

Created By: Module EDR

Notes:

1. This relation is used by module OFPEDR for output printing and punching.

Entity: **EPOINT**

Entity Type: Relation

Description: Contains the identification numbers of those points to be used as extra points. Input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Extra point set identification number
EXTID	Integer > 0	Extra point identification number

Created By: Module IFP

Entity: **FARGLIST**

Entity Type: Relation

Description: Contains the user function argument list information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
FINDX	Integer > 0	User function index number
ARGREF	Integer	Argument reference
IDENT	Text(8)	Internal user function name

Created By: Module FPKEVL

Entity: **FCHAIN**

Entity Type: Relation

Description: Contains the user function chain evaluation data generated from a lexical prepass.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
FINDX	Integer > 0	User function index number
IDENT	Text(8)	Internal user function name
NUMTOK	Integer	Number of tokens
TOKREF	Integer	Token reference in FTOKLIST and FDERLIST
NUMARG	Integer	Number of arguments
ARGREF	Integer	Argument reference in FARGLIST

Created By: Module FPKEVL

Entity: **FDERLIST**

Entity Type: Relation

Description: Contains the user function packet token list for function gradient evaluation.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
FINDX	Integer > 0	User function index number
TOKREF	Integer	Token reference
BAKREF	Integer	Back token reference
TOKEN	Text(8)	Token
INSCOD	Text(8)	Instruction code
OPCODE	Integer	Operator code
OPCOMP	Integer	Operator component
INTVAL	Integer	Integer value
RSPVAL	Real	Real value

Created By: Module FPKEVL

Entity: **FDAMPDVA**

Entity Type: Relation

Description: Contains the user function requested flutter damping response sensitivity information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
MXCOL	Integer > 0	Matrix column number in **[MXFDPDVA]** for the sensitivity values

Created By: Module FLUTSENS

Entity: **FDAMPRSP**

Entity Type: Relation

Description: Contains the user function requested flutter damping response values.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
VALUE	Real	Response value

Created By: Module FLUTTRAN

Entity: **FFREQDVA**

Entity Type: Relation

Description: Contains the user function requested flutter frequency response sensitivity information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
MXCOL	Integer > 0	Matrix column number in **[MXFFQDVA]** for the sensitivity values

Created By: Module FLUTSENS

Entity: **FFREQRSP**

Entity Type: Relation

Description: Contains the user function requested flutter frequency response values.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
VALUE	Real	Response value

Created By: Module FLUTTRAN

Entity: **FLCFDRVA**

Entity Type: Relation

Description: Contains the user function requested steady aeroelastic flexible stability coefficient response sensitivity information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
MXCOL	Integer > 0	Matrix column number in **[MXFCFDVA]** for the sensitivity values

Created By: Module AEROEEFS

Entity: **FLCFRESP**

Entity Type: Relation

Description: Contains the user function requested steady aeroelastic flexible stability coefficient response values.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
VALUE	Real	Response value

Created By: Module SAERO

Entity: **FFT**

Entity Type: Relation

Description: Contains the parameters required for controlling the Fast Fourier Transformation as input from the Bulk Data file

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer > 0, key	FFT set identification number
TIME	Real > 0	Length of time period
NT	Integer ≥ 2	Number of time points
RDELTF	Real	Ratio of frequency range increment to 1/TIME
RF	Real	Ratio of frequency range to NT/2*TIME
FRIM	Text (8)	Frequency interpolation method
OTYPE	Text (8)	Types of response output
FLIM	Text (8)	Frequency load interpolation method

Created By: Module IFP

Entity: **FLFACT**

Entity Type: Relation

Description: Contains flutter aerodynamic input data as defined on the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
VALUE	Real	Data value

Created By: Module IFP

Notes:

1. This relation contains one tuple for each value in each set defined on the FLFACT entry.

Entity: **FLUTMODE**

Entity Type: Matrix

Description: A matrix used to store the complex modal participation factors for any flutter eigenvectors computed during flutter analyses in analysis boundary conditions.

Matrix Form: A complex rectangular matrix with one row for each normal mode (including those omitted from the flutter analysis) and one column for each flutter eigenvector found in the current boundary condition.

Created By: FLUTTRAN

Notes:

1. The FLUTREL entity contains additional data to identify the flutter condition for each mode.
2. This entity is flushed between each analysis boundary condition having flutter analyses.
3. This entity is not used in the optimization boundary conditions.

Entity: **FLUTREL**

Entity Type: Relation

Description: Contains the flutter results for each flutter eigenvector/eigenvalue found during flutter analyses.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SUBID	Integer > 0	Flutter subcase identification number
COLUMN	Integer > 0	Column number in FLUTMODE for corresponding participation factors
MACH	Real	Flutter Mach number
RHORATIO	Real > 0	Flutter density ratio
RFRQ	Real > 0	Flutter reduced frequency
VEL	Real	Flutter velocity
RHOREF	Real > 0	Reference density
REFCHORD	Real > 0	Reference chord length

Created By: Module FLUTTRAN

Notes:

1. This entity is used to print the flutter mode shapes in physical coordinates.

Entity: **FLUTTER**

Entity Type: Relation

Description: Contains the definition of data needed to perform flutter analyses as input from the bulk data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
METHOD	Text (4)	Flutter analysis method = K, PK or KE
DENS	Integer > 0	Identification of FLFACT tuples defining density ratios
MACHVAL	Real > 0.0	Mach number to be used in the Flutter analyses
VEL	Integer > 0	Identification of FLFACT tuples specifying velocities
MLIST	Integer	Identification number of SET1 entries listing the normal modes to be omitted in the flutter analysis
KLIST	Integer	Identification of FLFACT tuples specifying a list of "hard point" reduced frequencies for the given Mach number for use in the Flutter analysis
EFFID	Integer	Identification of a CONEFFF set specifying control surface effectiveness values
SYMXZ	Integer	Symmetry flag for xz-plane
SYMXY	Integer	Symmetry flag for xy-plane
EPS	Real	Convergence parameter for flutter eigenvalue
CURVFIT	Text(8)	Type of curve fit to be used in the PK flutter analysis

Created By: Module IFP

Entity: **FNARO**

Entity Type: Relation

Description: Contains the user function instances requesting steady aeroelastic response information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DCREF	Integer > 0	User function set identification number
INSTANCE	Integer > 0	User function instance number
POSITION	Integer Vector (2)	Response position index in the user function
CLASS	Text (16)	User function class (OBJECTIVE, CONSTRAINT)
BCND	Integer > 0	Boundary condition identification number
CASEID	Integer > 0	Subcase identification number
TYPE	Text(8)	Response type (FLEXCF, RIGIDCF, TRIM)
AXIS	Text(8)	Rigid body axis of coefficient
PARAM	Text(8)	Trim parameter

Created By: Module FPKEVL

Notes:

1. The allowable values for AXIS are:
   DRAG, SIDE, LIFT, ROLL, PITCH, YAW.
2. The allowable values for PARAM are:
   ALPHA, BETA, PRATE, QRATE, RRATE, PACCEL, QACCEL, RACCEL, and user defined surfaces.

Entity: **FNCEN**

Entity Type: Relation

Description: Contains the user function instances requesting element centroidal coordinates information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DCREF	Integer > 0	User function set identification number
INSTANCE	Integer > 0	User function instance number
POSITION	Integer Vector (2)	Response position index in the user function
CLASS	Text (16)	User function class (OBJECTIVE, CONSTRAINT)
EID	Integer > 0	Element identification number
TYPE	Text(8)	Element type
COMP	Text(4)	Coordinate component (X1, X2, X3)
CID	Integer	Coordinate system identification number

Created By: Module FPKEVL

Entity: **FNCOR**

Entity Type: Relation

Description: Contains the user function instances requesting grid point coordinates information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DCREF	Integer > 0	User function set identification number
INSTANCE	Integer > 0	User function instance number
POSITION	Integer Vector (2)	Response position index in the user function
CLASS	Text (16)	User function class (OBJECTIVE, CONSTRAINT)
GID	Integer > 0	Grid point identification number
CID	Integer	Coordinate system identification number
COMP	Text(4)	Coordinate component (X1, X2, X3)

Created By: Module FPKEVL

Entity: **FNDNV**

Entity Type: Relation

Description: Contains the user function instances requesting global design variable information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DCREF	Integer > 0	User function set identification number
INSTANCE	Integer > 0	User function instance number
POSITION	Integer Vector (2)	Response position index in the user function
CLASS	Text (16)	User function clasee (OBJECTIVE, CONSTRAINT)
DV	Integer > 0	Design variable identification number

Created By: Module FPKEVL

Entity: **FNELM**

Entity Type: Relation

Description: Contains the user function instances requesting element stress and strain response information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DCREF	Integer > 0	User function set identification number
INSTANCE	Integer > 0	User function instance number
POSITION	Integer Vector (2)	Response position index in the user function
CLASS	Text (16)	User function class (OBJECTIVE, CONSTRAINT)
BCND	Integer > 0	Boundary condition identification number
DTYPE	Text(8)	Discipline type (STATICS, SAERO)
CASEID	Integer > 0	Subcase identification number
MODEID	Integer > 0	Mode number
EID	Integer > 0	Element identification number
TYPE	Text(8)	Element type
PLYID	Integer ≥ 0	Composite element layer number
QUANTY	Text(8)	Response type (STRESS, STRAIN)
COMP	Text(8)	Stress/strain component

Created By: Module FPKEVL

Notes:

1. The allowable stress components are:

ROD	SIGAXL, SIGTOR, SIG1, SIG2, MAXSHEAR
BAR	SIGAXL, SIGCA, SIGDA, SIGEA, SIGFA, SIGCB, SIGDB, SIGEB, SIGFB
SHEAR	MAXSHEAR
QDMEM1 TRMEM	SIGX, SIGY, TAUXY, SIG1, SIG2, MAXSHEAR, FIBER, TRANSV
QUAD4 TRIA3	SIGX, SIGY, TAUXY, SIG1, SIG2, MAXSHEAR, FIBER, TRANSV,TSIGX, TSIGY, TTAUXY, TSIG1, TSIG2, TMAXSHEAR, TFIBER, TTRANSV, BSIGX, BSIGY, BTAUXY, BSIG1, BSIG2, BMAXSHEAR, BFIBER, BTRANSV

2. The allowable strain components are:

ROD	EPSAXL, EPSTOR, EPS1, EPS2, MAXSHEAR
BAR	EPSAXL, EPSCA, EPSDA, EPSEA, EPSFA, EPSCB, EPSDB, EPSEB, EPSFB
SHEAR	MAXSHEAR
QDMEM1 TRMEM	EPSX, EPSY, TAUXY, EPS1, EPS2, MAXSHEAR, FIBER, TRANSV
QUAD4 TRIA3	EPSX, EPSY, TAUXY, EPS1, EPS2, MAXSHEAR, FIBER, TRANSV,TEPSX, TEPSY, TTAUXY, TEPS1, TEPS2, TMAXSHEAR, TFIBER, TTRANSV, BEPSX, BEPSY, BTAUXY, BEPS1, BEPS2, BMAXSHEAR, BFIBER, BTRANSV

Entity: **FNFLT**

Entity Type: Relation

Description: Contains the user function instances requesting flutter analysis response information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DCREF	Integer > 0	User function set identification number
INSTANCE	Integer > 0	User function instance number
POSITION	Integer Vector (2)	Response position index in the user function
CLASS	Text (16)	User function class (OBJECTIVE, CONSTRAINT)
BCND	Integer > 0	Boundary condition identification number
CASEID	Integer > 0	Subcase identification number
MODEID	Integer > 0	Mode number
MACH	Real > 0.0	Mach value
DENSITY	Real > 0.0	Density value
VELOCITY	Real > 0.0	Velocity value
QUANTY	Text(8)	Response type (FROOT, FDAMP, FREQ)
COMP	Text(8)	Damping component

Created By: Module FPKEVL

Entity: **FNFRQ**

Entity Type: Relation

Description: Contains the user function instances requesting normal mode frequency response information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DCREF	Integer > 0	User function set identification number
INSTANCE	Integer > 0	User function instance number
POSITION	Integer Vector (2)	Response position index in the user function
CLASS	Text (16)	User function class (OBJECTIVE, CONSTRAINT)
BCND	Integer > 0	Boundary condition identification number
CASEID	Integer > 0	Subcase identification number
MODEID	Integer > 0	Mode number

Created By: Module FPKEVL

Entity: **FNGRD**

Entity Type: Relation

Description: Contains the user function instances requesting grid point displacement response information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DCREF	Integer > 0	User function set identification number
INSTANCE	Integer > 0	User function instance number
POSITION	Integer Vector (2)	Response position index in the user function
CLASS	Text (16)	User function class (OBJECTIVE, CONSTRAINT)
BCND	Integer > 0	Boundary condition identification number
DTYPE	Text(8)	Discipline type (STATICS, SAERO)
CASEID	Integer > 0	Subcase identification number
MODEID	Integer > 0	Mode number
GID	Integer > 0	Grid point identification number
CID	Integer	Coordinate system identification number
QUANTY	Text(8)	Response type (DISP)
COMP	Text(8)	Displacement component (T1, T2, T3, R1, R2, R3)

Created By: Module FPKEVL

Entity: **FNMAS**

Entity Type: Relation

Description: Contains the user function instances requesting element mass information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DCREF	Integer > 0	User function set identification number
INSTANCE	Integer > 0	User function instance number
POSITION	Integer Vector (2)	Response position index in the user function
CLASS	Text (16)	User function class (OBJECTIVE, CONSTRAINT)
EID	Integer > 0	Element identification number
TYPE	Text(8)	Element type
PLYID	Integer ≥ 0	Composite element layer number

Created By: Module FPKEVL

Entity: **FNTHK**

Entity Type: Relation

Description: Contains the user function instances requesting element thickness information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DCREF	Integer > 0	User function set identification number
INSTANCE	Integer > 0	User function instance number
POSITION	Integer Vector (2)	Response position index in the user function
CLASS	Text (16)	User function class (OBJECTIVE, CONSTRAINT)
EID	Integer > 0	Element identification number
TYPE	Text(8)	Element type
PLYID	Integer ≥ 0	Composite element layer number

Created By: Module FPKEVL

Entity: **FNWGH**

Entity Type: Relation

Description: Contains the user function instances requesting element weight information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DCREF	Integer > 0	User function set identification number
INSTANCE	Integer > 0	User function instance number
POSITION	Integer Vector (2)	Response position index in the user function
CLASS	Text (16)	User function class (OBJECTIVE, CONSTRAINT)
EID	Integer > 0	Element identification number
TYPE	Text(8)	Element type
PLYID	Integer ≥ 0	Composite element layer number

Created By: Module FPKEVL

Entity: **FORCE**

Entity Type: Relation

Description: Contains the definition of a static load at a grid point as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
GRID1	Integer > 0	Grid point at which the load is applied
CID1	Integer ≥ 0	Coordinate system identification
SCALE	Real	Scale factor
N1, N2, N3	Real	Components of the force vector

Created By: Module IFP

Entity: **FORCE1**

Entity Type: Relation

Description: Contains the definition of a load applied at a grid point with the direction determined by a line connecting two grid points.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
GRID1	Integer > 0	Grid point id at which the force is applied
SCALE	Real	Scale factor
GRID2, GRID3	Integer > 0	Grid point identification

Created By: Module IFP

Entity: **FREQ**

Entity Type: Relation

Description: Contains frequency values to be used for solution in frequency response.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer > 0	Set identification number
FREQ	Real	Frequency value

Created By: Module IFP

Notes:

1. The relation is used in subroutine PREFRQ in the generation of the FREQL entity.
2. The unit for FREQ is Hertz.
3. The set is selected in Solution Control.

Entity: **FREQ1**

Entity Type: Relation

Description: Contains information to specify frequencies used in frequency response solution as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer > 0	Set identification number
F1	Real ≥ 0.0	First frequency in a set
DFRQ	Real > 0.0	Frequency increment
NDFR	Integer	Number of increments

Created By: Module IFP

Notes:

1. The relation is used in subroutine PREFRQ in the generation of the FREQL entity.
2. Units for F1 and DFREQ, when input, are Hertz.
3. The set is selected in Solution Control.

Entity: **FREQ2**

Entity Type: Relation

Description: Contains information to specify frequencies used in frequency response solution as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer > 0	Set identification number
F1	Real > 0.0	First frequency value
F2	Real > 0.0, F2 > F1	Last frequency value
NLOGI	Integer > 0	Number of increments

Created By: Module IFP

Notes:

1. The relation is used in subroutine PREFRQ in the generation of the FREQL entity.
2. Units for F1 and F2, when input, are Hertz.
3. The set is selected in Solution Control.

Entity: **FREQL**

Entity Type: Unstructured

Description: Contains a list of frequencies for each frequency set.

Record:

1. Contains a list of the LIDs of the frequency sets in the Bulk Data file.
i. Contains the frequency list for the $(i-1)^{th}$ set ID.

Created By: Module PFBULK

Notes:

1. This entity is used in the generation of frequency dependent loads in the DMA module.

Entity: **FREQLIST**

Entity Type: Relation

Description: Contains the list of frequencies for which outputs are requested as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer	Set identification number
FREQ	Real	Frequency value in Hertz.

Created By: Mobile IFP

Entity: **FREQDRVA**

Entity Type: Relation

Description: Contains the user function requested normal mode response sensitivity information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
MXCOL	Integer > 0	Matrix column number in [MXFRQDVA] for the sensitivity values

Created By: Module FREQSENS

Entity: **FREQRESP**

Entity Type: Relation

Description: Contains the user function requested normal mode response values.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
VALUE	Real	Response value

Created By: Module FCEVAL

Entity: **FROOTDVA**

Entity Type: Relation

Description: Contains the user function requested flutter root response sensitivity information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
MXCOL	Integer > 0	Matrix column number in [MXFRTDVA] for the sensitivity values

Created By: Module FLUTSENS

Entity: **FROOTRSP**

Entity Type: Relation

Description: Contains the user function requested flutter root response values.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
RVALUE	Real	Real part of the response value
IVALUE	Real	Imaginary part of the response value

Created By: Module FLUTTRAN

Entity: **FTF**

Entity Type: Matrix

Description: A matrix used in the nuclear blast response calculation to compute the forces on structural modes given the forces at the aerodynamic panels.

Matrix Form: A real, rectangular matrix with the number of rows equal to the number of aerodynamic panels.

Created By: MAPOL

Notes:

1. FTF is computed using:

$$[FTF] = [PHIB]^T * [BLGTJA]$$

Entity: **FTOKLIST**

Entity Type: Relation

Description: Contains the user function packet token list for function evaluation.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
FINDX	Integer > 0	User function index number
TOKREF	Integer	Token reference
TOKEN	Text(8)	Token
INSCOD	Text(8)	Instruction code
OPCODE	Integer	Operator code
OPCOMP	Integer	Operator component
INTVAL	Integer	Integer value
RSPVAL	Real	Real value

Created By: Module FPKEVL

Entity: **GASUBO**

Entity Type: Subscripted Matrix

Description: Contains the matrix product:

$$- [KAOO]^{-1} [KAOA]$$

used in the static reduction of the free degrees of freedom. This matrix includes the aeroelastic terms.

Matrix Form: A variable-sized matrix having one row for each omitted degree of freedom and one column for each degree of freedom in the analysis set for the current boundary condition.

Created By: MAPOL

Notes:

1. The dimension of this subscripted matrix must be large enough for all optimization and analysis boundary conditions.

Entity: **GDVLIST**

Entity Type: Relation

Description: Contains the definition of the list of global design variables as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer	Set identification number
DVID	Integer	Global design variable identification

Created By: Module IFP

Entity: **GDVDRVA**

Entity Type: Relation

Description: Contains the user function requested global design variable sensitivity information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
MXCOL	Integer > 0	Matrix column number in [MXGDVDVA] for the sensitivity values

Created By: Module GRVGRAD

Entity: **GDVRESP**

Entity Type: Relation

Description: Contains the user function requested global design variable values.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
VALUE	Real	Response value

Created By: Module GDVRESP

Entity: **GENEL**

Entity Type: Unstructured

Description: Contains GENEL data as input from the Bulk Data file

Entity Structure: One record for each GENEL entry. Each record has the following structure:

WORD	NAME	DESCRIPTION
1	EID	Element identification number
2	NGI	Number of GIDI/COMPI pairs
3	NGD	Number of GIDD/COMPD pairs
4	NKZ	Number of NK or Z terms
5	" K" or " Z"	(Hollerith indicating K or Z)
6	NS	
7 to 6+2 NGI	GIDI COMPI	List of u_i grid/components. The NGI pairs are the GIDI/COMPI values (2 NGI words)
7+2 to 6+2 NGI +2 NGD	GIDD COMPD	List of u_d grid/components. The NGD pairs are the GIDD/COMPD values (2 NGD words)
7+2 NGI +2 NGD to 6+2 NGI +2 NGD + NKZ	" K" or " Z"	List of the NKZ terms of the "K" or "Z" matrix
7+2 NGI +2 NGD + NKZ to END	S	List of the NS terms of the S matrix

Any of NGI, NGD, NKZ or NS can be zero with the resultant list being absent from the record. The order of terms will be maintained as per the input entry.

Created By: Module IFP

Entity: **GENFA**

Entity Type: Matrix

Description: The generalized force matrix form of the MATSS matrix developed for nuclear blast response.

Matrix Form: Real rectangular matrix with the number of rows equal to the number of modes retained for the blast analysis and the number of columns equal to the number of panels in the unsteady aerodynamics model.

Created By: Module MAPOL

Entity:	**GENF**
Entity Type:	Matrix
Description:	The generalized matrix form of the BFRC matrix developed for nuclear blast response.
Matrix Form:	Real rectangular matrix with the number of rows equal to the number of modes retained for the blast analysis and three columns.
Created By:	Module MAPOL

Entity:	**GENK**
Entity Type:	Matrix
Description:	The generalized matrix form of the KAA matrix developed for nuclear blast response.
Matrix Form:	Real square matrix with the number of rows equal to the number of modes retained for the blast analysis.
Created By:	Module MAPOL

Entity:	**GENM**
Entity Type:	Matrix
Description:	The generalized matrix form of the MAA matrix developed for nuclear blast response.
Matrix Form:	Real square matrix with the number of rows and columns equal to the number of modes retained for the blast analysis.
Created By:	Module MAPOL

Entity:	**GENQL**
Entity Type:	Matrix
Description:	The generalized matrix form of the MATTR matrix developed for nuclear blast response.
Matrix Form:	Real rectangular matrix with the number of rows equal to the number of modes retained for the blast analysis and the number of columns equal to the number of panels in the unsteady aerodynamics model times the number of beta values used in the fitting process.
Created By:	Module MAPOL

Entity:	**GENQ**
Entity Type:	Matrix
Description:	The generalized matrix form of the MATSS matrix developed for nuclear blast response.
Matrix Form:	Real square matrix with the number of rows equal to the number of modes retained for the blast analysis and the number of columns equal to the number of panels in the unsteady aerodynamics model times the number of beta values used in the fitting process.
Created By:	Module MAPOL

Entity:	**GFE**
Entity Type:	Matrix
Description:	A partition of the GENF matrix (see GENF).

Entity:	**GFR**
Entity Type:	Matrix
Description:	A partition of the GENF matrix (see GENF).

Entity: **GEOMSA**

Entity Type: Relation

Description: Contains data on the geometric location of the aerodynamic degrees of freedom for the planar and nonplanar steady aerodynamics models.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
MODEL	Integer	Planar or nonplanar steady aerodynamics model identifier = 1 for planar model = – 2 for nonplanar model
MACROID	Integer	Component identification number
ACMPNT	Text (8)	Component type One of WING, FIN, CANARD, POD, or FUSEL
NDOF	Integer	Number of degrees of freedom at the point. = 1 for all steady aerodynamic boxes refer to GEOMUA for unsteady model options.
EXTID	Integer	External box identification number
INTID	Integer	Internal box identification number. This is the row and/or column number in the AICMAT
AREA	Real	The area of the box.
X	Real	The x location of the box centroid in basic coordinates.
Y	Real	The y location of the box centroid in basic coordinates.
Z	Real	The z location of the box centroid in basic coordinates.
N1	Real	The x component of the box normal in basic coordinates.
N2	Real	The y component of the box normal in basic coordinates.
N3	Real	The z component of the box normal in basic coordinates.
R1	Real	The x component of the box rotation axis in basic coordinates.
R2	Real	The y component of the box rotation axis in basic coordinates.
R3	Real	The z component of the box rotation axis in basic coordinates.

Created By: Module STEADY and STEADYNP

Notes:

1. These data are used in splining the aerodynamic forces to the structural model, in splining structural deflections to the aerodynamic model and in recovering trimmed pressures and displacements on the aerodynamic model.

Entity: **GEOMUA**

Entity Type: Relation

Description: Contains data on the geometric location of the aerodynamic degrees of freedom for the unsteady aerodynamics models. Two entries are loaded for ZY boxes to account for the two normals during splining.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
MACROID	Integer	Component identification number
ACMPNT	Text (8)	Component type One of WING, FIN, CANARD, POD, or FUSEL
NDOF	Integer	Number of degrees of freedom at the point. = 1 for all lifting surface boxes CAERO1 = 2 for all body surface boxes CAERO2
EXTID	Integer	External box identification number
INTID	Integer	Internal box identification number.
AREA	Real	The area of the box.
X	Real	The x location of the box centroid in basic coordinates.
Y	Real	The y location of the box centroid in basic coordinates.
Z	Real	The z location of the box centroid in basic coordinates.
N1	Real	The x component of the box normal in basic coordinates.
N2	Real	The y component of the box normal in basic coordinates.
N3	Real	The z component of the box normal in basic coordinates.
R1	Real	The x component of the box rotation axis in basic coordinates.
R2	Real	The y component of the box rotation axis in basic coordinates.
R3	Real	The z component of the box rotation axis in basic coordinates.

Created By: Module UNSTEADY

Entity: **GGO**

Entity Type: Matrix

Description: Rigid body transformation matrix to transfer displacements at the origin of the basic coordinate system to g-set displacements.

Matrix Form: Real rectangular matrix with g-set rows and up to six columns.

Created By: Module GDR1

Entity: **GLBDES**

Entity Type: Relation

Description: Contains current design variable information for all design variables in the problem.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
NITER	Integer	Iteration number
DVID	Integer > 0, key	Design variable identification number
OPTION	Integer 1, 2, 3	Design variable linking option
LINKID	Integer	Link set identification
EID	Integer > 0	Element id if design variable uniquely linked to one element
ETYPE	Text (8)	Element type if unique linking
LAYRNUM	Integer	Layer number if unique or physical linking
LAYRLST	Integer	PLYLIST identifier if multiple plies are linked together
DVSYMBL	Text(8)	Designed dimension symbol (D1, D2, D3, D4, D5, D6, D7, D8, D9, D10 for BAR element cross section dimension; A for element area, T for element thickness; M for element mass; K for element stiffness.)
VMIN	Real	Minimum value for design variable
VMAX	Real	Maximum value for design variable
VALUE	Real	Current value of design variable
LABEL	Text (8)	User identification label

Created By: Module MAKEST

Notes:

1. The linking options are:
   OPTION = 1 DESELM linking
   OPTION = 2 DESVARP linking
   OPTION = 3 DESVARS linking
2. Design variable offset is stored in INFO(11) from MAKEST. The value is transferred to DVCT in the EMG module and DVCTD, DDVCT in the NLEMG module. The offset is used to generate pseudo design variables to control the bending behavior of designed CBAR elements.

Entity: **GLBSIG**

Entity Type: Matrix

Description: Contains the stress and strain components in the element coordinate system for linearly designed elements constrained through stress/strain constraint bulk data entries.

Matrix Form: A variable-size matrix having one row for each stress/strain component for each element subject to a linear strength constraint and one column for each load condition within each boundary condition. The order of the matrix rows is in element id order of linearly constrained elements within each element type. The element types are currently processed in the following order:

(1) BAR; $\sigma_{a1}, \sigma_{a2}, \sigma_{a3}, \sigma_{a4}, \sigma_{b1}, \sigma_{b2}, \sigma_{b3}, \sigma_{b4}$
(2) QDMEM1; $\sigma_x, \sigma_y, \tau_{xy}$
(3) QUAD4; $\sigma_x, \sigma_y, \tau_{xy}$
(4) ROD; σ_x, τ_{xy}
(5) SHEAR; τ_{xy}
(6) TRIA3; $\sigma_x, \sigma_y, \tau_{xy}$
(7) TRMEM; $\sigma_x, \sigma_y, \tau_{xy}$

The columns are processed in load condition order for each boundary condition.

Created By: Module SCEVAL

Notes:

1. If no linearly designed elements are constrained, this matrix will be empty.
2. Refer to the SMAT documentation for further details as GLBSIG is essentially:

   $[SMAT]^t [u_g]$

3. Each boundary condition's load conditions are appended onto the existing GLBSIG columns within the SCEVAL module.

Entity: **GMKCT**

Entity Type: Relation

Description: Contains data required to interpret the DKVI unstructured entity. The relation is sorted by DVID and KSIL.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DVID	Integer ≥ 0	Design variable identification
KSIL	Integer > 0	Internal id of grid or scalar point connected to the design variable
KCODE	Integer > 0	Codeword denoting the form in which the DKVI data are stored
NODES	Integer 0 < NODES ≤ 20	Number of nodes being processed with this KSIL
IREC	Integer > 0	Record number of DKVI entity where data are stored
ASIL	Ivector (20)	List of associated SILS
ALPHA	Real	Exponential power associated with BAR element design variable

Created By: Module NLEMA1

Notes:

1. KCODE definitions:
   = 1 Multiple associated grids with both extensional and rotational degrees of freedom.

   = 2 Column being assembled is a scalar point. Associated SILS may or may not be scalar points.
   = 3 Column being assembled is a grid point. At least one row is a scalar point.
   = 4 Only extensional degrees of freedom are included in the assembly process
2. The INFO array for this entity contains additional memory management allocation data:
   INFO(11) The number of tuples connected to the zeroth design variable.
   INFO(12) The maximum number of tuples connected to any one of the remaining design variables.

Entity: **GMKCT0**

Entity Type: Relation

Description: Contains data required to interpret the DKVI0 unstructured entity. The relation is sorted by DVID and KSIL.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DVID	Integer > 0	Design variable identification
KSIL	Integer > 0	Internal id of grid or scalar point connected to the design variable
KCODE	Integer > 0	Codeword denoting the form in which the DKVI0 data are stored
NODES	Integer 0 < NODES ≤ 20	Number of nodes being processed with this KSIL
IREC	Integer > 0	Record number of DKVI0 entity where data are stored
ASIL	Ivector (20)	List of associated SILS
ALPHA	Real	Exponential power associated with BAR element design variable

Created By: Module EMA1

Notes:

1. KCODE definitions:
   = 1 Multiple associated grids with both extensional and rotational degrees of freedom.

   = 2 Column being assembled is a scalar point. Associated SILS may or may not be scalar points.
   = 3 Column being assembled is a grid point. At least one row is a scalar point.
   = 4 Only extensional degrees of freedom are included in the assembly process
2. The INFO array for this entity contains additional memory management allocation data:
   INFO(11) The number of tuples connected to the zeroth design variable.
   INFO(12) The maximum number of tuples connected to any one of the remaining design variables.

Entity: **GMKCTG**

Entity Type: Relation

Description: Contains data required to interpret the DKVIG unstructured entity. The relation is sorted by DVID and KSIL.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DVID	Integer > 0	Design variable identification
KSIL	Integer > 0	Internal id of grid or scalar point connected to the design variable
KCODE	Integer > 0	Codeword denoting the form in which the DKVIG data are stored
NODES	Integer 0 < NODES ≤ 20	Number of nodes being processed with this KSIL
IREC	Integer > 0	Record number of DKVIG entity where data are stored
ASIL	Ivector (20)	List of associated SILS
ALPHA	Real	Exponential power associated with BAR element design variable

Created By: Module NLEMA1

Notes:

1. KCODE definitions:
   - = 1 Multiple associated grids with both extensional and rotational degrees of freedom.
   - = 2 Column being assembled is a scalar point. Associated SILS may or may not be scalar points.
   - = 3 Column being assembled is a grid point. At least one row is a scalar point.
   - = 4 Only extensional degrees of freedom are included in the assembly process
2. The INFO array for this entity contains additional memory management allocation data:
   - INFO(11) The number of tuples connected to the zeroth design variable.
   - INFO(12) The maximum number of tuples connected to any one of the remaining design variables.

Entity: **GMMCT**

Entity Type: Relation

Description: Contains data required to interpret the DMVI unstructured entity. The relation is sorted by DVID and KSIL.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DVID	Integer > 0	Design variable identification
KSIL	Integer > 0	Internal id of grid or scalar point connected to the design variable
MCODE	Integer > 0	Codeword denoting the form in which the DMVI data are stored
NODES	Integer 0 < NODES ≤ 20	Number of nodes being processed with this KSIL
IREC	Integer > 0	Record number of DMVI entity where data are stored
ASIL	Ivector (20)	List of associated SILS
ALPHA	Real	Exponential power associated with BAR element design variable

Created By: Module NLEMA1

Notes:

1. MCODE definitions:
   - = 1 Multiple associated grids with both extensional and rotational degrees of freedom.
   - = 2 Column being assembled is a scalar point. Associated SILS may or may not be scalar points.
   - = 3 Column being assembled is a grid point. At least one row is a scalar point.
   - = 4 Only extensional degrees of freedom are included in the assembly process.
2. The INFO array for this entity contains additional memory management allocation data:
   - INFO(11) The number of tuples connected to the zeroth design variable.
   - INFO(12) The maximum number of tuples connected to any one of the remaining design variables.

Entity: **GMMCT0**

Entity Type: Relation

Description: Contains data required to interpret the DMVI0 unstructured entity. The relation is sorted by DVID and KSIL.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DVID	Integer > 0	Design variable identification
KSIL	Integer > 0	Internal id of grid or scalar point connected to the design variable
MCODE	Integer > 0	Codeword denoting the form in which the DMVI0 data are stored
NODES	Integer 0 < NODES ≤ 20	Number of nodes being processed with this KSIL
IREC	Integer > 0	Record number of DMVI0 entity where data are stored
ASIL	Ivector (20)	List of associated SILS
ALPHA	Real	Exponential power associated with BAR element design variable

Created By: Module EMA1

Notes:

1. MCODE definitions:
   - = 1 Multiple associated grids with both extensional and rotational degrees of freedom.
   - = 2 Column being assembled is a scalar point. Associated SILS may or may not be scalar points.
   - = 3 Column being assembled is a grid point. At least one row is a scalar point.
   - = 4 Only extensional degrees of freedom are included in the assembly process.
2. The INFO array for this entity contains additional memory management allocation data:
   - INFO(11) The number of tuples connected to the zeroth design variable.
   - INFO(12) The maximum number of tuples connected to any one of the remaining design variables.

Entity: **GMMCTD**

Entity Type: Relation

Description: Contains data required to interpret the DMVID unstructured entity. The relation is sorted by DVID and KSIL.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DVID	Integer > 0	Design variable identification
KSIL	Integer > 0	Internal id of grid or scalar point connected to the design variable
MCODE	Integer > 0	Codeword denoting the form in which the DMVID data are stored
NODES	Integer 0 < NODES ≤ 20	Number of nodes being processed with this KSIL
IREC	Integer > 0	Record number of DMVID entity where data are stored
ASIL	Ivector (20)	List of associated SILS
ALPHA	Real	Exponential power associated with BAR element design variable

Created By: Module NLEMA1

Notes:

1. MCODE definitions:
   - = 1 Multiple associated grids with both extensional and rotational degrees of freedom.
   - = 2 Column being assembled is a scalar point. Associated SILS may or may not be scalar points.
   - = 3 Column being assembled is a grid point. At least one row is a scalar point.
   - = 4 Only extensional degrees of freedom are included in the assembly process.
2. The INFO array for this entity contains additional memory management allocation data:
   - INFO(11) The number of tuples connected to the zeroth design variable.
   - INFO(12) The maximum number of tuples connected to any one of the remaining design variables.

Entity: **GMMCTG**

Entity Type: Relation

Description: Contains data required to interpret the DMVIG unstructured entity. The relation is sorted by DVID and KSIL.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
DVID	Integer > 0	Design variable identification
KSIL	Integer > 0	Internal id of grid or scalar point connected to the design variable
MCODE	Integer > 0	Codeword denoting the form in which the DMVIG data are stored
NODES	Integer 0 < NODES ≤ 20	Number of nodes being processed with this KSIL
IREC	Integer > 0	Record number of DMVIG entity where data are stored
ASIL	Ivector (20)	List of associated SILS
ALPHA	Real	Exponential power associated with BAR element design variable

Created By: Module NLEMA1

Notes:

1. MCODE definitions:
   - = 1 Multiple associated grids with both extensional and rotational degrees of freedom.
   - = 2 Column being assembled is a scalar point. Associated SILS may or may not be scalar points.
   - = 3 Column being assembled is a grid point. At least one row is a scalar point.
   - = 4 Only extensional degrees of freedom are included in the assembly process.
2. The INFO array for this entity contains additional memory management allocation data:
   - INFO(11) The number of tuples connected to the zeroth design variable.
   - INFO(12) The maximum number of tuples connected to any one of the remaining design variables.

Entity: **GPFDATA**

Entity Type: Relation

Description: Contains the grid point forces.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
BC	Integer > 0	Boundary condition identification number
NITER	Integer > 0	Iteration number for optimization
DISC	Integer > 0	Discipline type = 1 Statics = 5 Transient = 2 Modes = 7 Buckling = 3 Steady Aero = 8 Blast
SUBCASE	Integer > 0	Subcase identification number if (DISC = 1, 3, 5, 8) or Mode Number if (DISC = 2, 7)
EID	Integer > 0	Element identification number
ETYPE	Text(8)	Element type (example: "BAR")
CMPLX	Integer > 0	Complex output identifier = 1 if real response quantities = 2 if complex response quantities
SIL	Integer > 0	Internal grid point identification number
FLAG	Integer > 0	Flag indicating the point is a grid point or a scalar point
RFORCE	Real vector(6)	Real parts of force components
IFORCE	Real vector(6)	Imaginary parts of force components

Created By: Module EDR

Notes:

1. This relation is used by module OFPEDR for output printing and punching.
2. The FLAG equals 6 if the point is a grid point and equals 1 if a scalar point and 0 if not in the g-set.

Entity: **GPFELEM**

Entity Type: Relation

Description: Contains the list of elements connected to structural nodes for which grid point forces have been requested.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer > 0	Set identification number of GPFORCE print request
ETYPE	Text(8)	Element type (example: "BAR")
EID	Integer > 0	Element identification number
NGRID	Integer > 0	Number of grid points in the element for which grid point forces are needed
AGRID	Ivector (32)	Array containing sorted list of GIDs

Created By: Module PFBULK

Notes:

1. SID refers to the GRIDLIST bulk data entry used in the GPFORCE Solution Control print request id in Solution Control.

Entity: **GPST**

Entity Type: Unstructured

Description: Contains the grid point singularity summary table.

Entity Structure:

Record:

1. There are ten words for each grid point component or each scalar point, and all of the summary table are in one record.
   Each ten words are as follows:

WORD	DESCRIPTION
1	External ID
2	Type (GRID, SCALAR) G or S in hollerith
3	Component if grid point
4,5,6	Singularity direction vector
7	Stiffness ratio
8	Old set label
9	New set label
10	Asterisk or blank (Asterisk means USET updated)

Created By: GPSP

Entity: **GPWGGRID**

Entity Type: Relation

Description: Contains the definition of a location for grid point weight generation as input by the GPWG data entry in the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
GRIDPNT	Integer	Grid point identification or NULL if explicit location is given
X0	Real	x location of the point
Y0	Real	y location of the point
Z0	Real	z location of the point

Created By: Module IFP

Notes:

1. GRIDPNT and X0,Y0,Z0 are mutually exclusive mechanisms to enter a location
2. Only the first tuple of GPWGGRID will be used
3. If no tuples exist, the GPWG is performed at the origin of the basic coordinate system

Entity: **GRADIENT**

Entity Type: Relation

Description: Contains the gradients of objective and constraint gradients requested to be printed or punched.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
PRINTKEY	Integer	Key value referred to by the CONST relation
DVID	Integer	Design variable identification number
GRADIENT	Real	Gradient value for the constraint in CONST with the associated PRINTKEY

Created By: Module DESIGN or VANGO

Notes:

1. This entity contains one tuple for every global design variable for each tuple of CONST with a nonzero PRINTKEY attribute. That attribute points to the associated tuples of this entity.

Entity: **GRAV**

Entity Type: Relation

Description: Contains the definition of the gravity vectors to be used in applying gravity loading to the model.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
CID1	Integer ≥ 0	Coordinate system id
SCALE	Real	Scale factor
N1, N2, N3	Real	Components of the gravity vector

Created By: Module IFP

Entity: **GRID**

Entity Type: Relation

Description: Contains the geometric and permanent constraint data for a structural grid point as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
GRIDID	Integer > 0, key	The external grid point id
CP	Integer ≥ 0	The coordinate system in which the location of the point is defined
X, Y, Z	Real	The location of the grid point
CD	Integer ≥ 0	The id of the coordinate system to be used to define displacements, constraints, degrees of freedom, and solution vectors
PERMSPC	Integer ≥ 0	The permanent single point constraints associated with the grid point

Created By: Module IFP

Entity: **GRIDLIST**

Entity Type: Relation

Description: Contains the list of grid, scalar or extra points for which node dependent outputs are requested as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer	Set identification number
GID	Integer	External grid/scalar/extra point identification number

Created By: Module IFP

Entity: **GRIDTEMP**

Entity Type: Unstructured

Description: Contains temperature data for all grid and scalar points for all thermal load sets defined.

Record:

1. A list of all set identification numbers in sorted order.
i. Contains the grid and scalar point temperatures for the (i-1)th specified thermal load case. The storage order of temperature data within each record follows the ordering of KSIL values in the BGPDT table with extra points excluded.

Created By: PFBULK

Notes:

1. This entity is used in EMG to compute average element temperatures and is used in LODGEN to compute the global thermal load sensitivity vectors.

Entity: **GSKF**

Entity Type: Matrix

Description: The transpose of GSTKF (see GSTKG).

Entity: **GSTKF**

Entity Type: Matrix

Description: A partition of the GSTKN matrix (see GSTKG)

Entity: **GSTKG**

Entity Type: Matrix

Description: The interpolation matrix relating slopes in the streamwise direction at the aerodynamic degrees of freedom to the displacements at the global structural degrees of freedom.

Matrix Form: A variable-sized matrix having one column for each steady aerodynamic box and one row for each structural degree of freedom.

Created By: Module SPLINES

Notes:

1. The MAPOL sequence supports the following partitions of the GSTKG matrix (see the Theoretical Manual for the explicit formation of these submatrices):

$$\mathbf{GSTKG} \rightarrow \begin{bmatrix} \varphi \\ \mathbf{GSTKN} \end{bmatrix}$$

$$\mathbf{GSTKN} \rightarrow \begin{bmatrix} \varphi \\ \mathbf{GSTKF} \end{bmatrix}$$

$$\mathbf{GSKF} = [\mathbf{GSTKF}]^{\mathbf{T}}$$

Entity: **GSTKN**

Entity Type: Matrix

Description: A partition of the GSTKG matrix (see GSTKG).

Entity: **GSUBO**

Entity Type: Subscripted Matrix

Description: See Notes.

Matrix Form: A variable-sized matrix with one row for each omitted degree of freedom and one column for each degree of freedom retained for analysis. The precision of this matrix is the same as that of the KGG matrix.

Created By: MAPOL

Notes:

1. For Guyan reduction GSUBO contains the matrix product

   $$-\ [KOO]^{-1}\ [KOA]$$

2. If no omitted degrees of freedom are defined for the model, GSUBO will be initialized.
3. For generalized dynamic reduction, GSUBO relates degrees of freedom in the f- to q- set (union of a-, k-, and j-sets) degrees of freedom.

Entity: **GTKF**

Entity Type: Matrix

Description: A partition of the GTKN matrix (see GTKN).

Entity: **GTKG**

Entity Type: Matrix

Description: The interpolation matrix relating the forces at the aerodynamic degrees of freedom to the forces at the global structural degrees of freedom.

Matrix Form: A variable-sized matrix having one column for each steady aerodynamic degree of freedom (box) and one row for each structural degree of freedom.

Created By: Module SPLINES

Notes:

1. The MAPOL sequence creates the following partitions of the GTKG matrix (see the Theoretical Manual for the explicit formation of these submatrices):

   $$GTKG \rightarrow \begin{bmatrix} \varphi \\ GTKN \end{bmatrix}$$

   $$GTKN \rightarrow \begin{bmatrix} \varphi \\ GTKF \end{bmatrix}$$

Entity: **GTKN**

Entity Type: Matrix

Description: A partition of the GTKG matrix (see GTKG).

Entity: **GUST**

Entity Type: Relation

Description: Contains vertical gust data for a gust analysis as input from the bulk data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer > 0	Gust set ID
GLOAD	Integer > 0	ID of an entry which defines time or frequency dependent loads.
WG	Real > 0.0	Scale factor for gust velocity
XO	Real ≥ 0.0	Location of reference plane in aerodynamic coordinates
V	Real > 0	Velocity of the vehicle
QDP	Real > 0	Dynamic pressure of the vehicle
MACH	Real ≥ 0.0	Mach number of the vehicle
SYMXZ	Integer	Symmetry flag for xz-plane
SYMXY	Integer	Symmetry flag for xy-lane

Created By: Module IFP

Entity: **IARGLIST**

Entity Type: Relation

Description: Contains the resolved user function argument list information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer > 0	User function instance number
FINDX	Integer > 0	User function index number
ARGREF	Integer	Argument reference
IDENT	Text(8)	Internal function name
TYPE	Integer	Argument type
INTVAL	Integer	Integer value
RSPVAL	Real	Real value

Created By: Module FPKEVL

Entity: **IC**

Entity Type: Relation

Description: Contains the values of initial displacements and velocities for use in direct transient response analysis as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer	Set identification number
GRID	Integer	Grid, scalar or extra point identification number
COMP	Integer	Grid point component number
U0	Real	Initial displacement value
V0	Real	Initial velocity value

Created By: Module IFP

Entity: **ICDATA**

Entity Type: Unstructured

Description: Contains the collection of initial condition data for all initial condition sets referenced in Solution Control.

Entity Structure:

Record:

1. A list of all set identification numbers in sorted order.

i. Contains the initial conditions for the $(i-1)^{th}$ initial condition set. Each record has the following form:

WORD	VARIABLE	DESCRIPTION
1	SID	Set identification
j	ROW	Internal number of the degree of freedom affected by the initial condition in increasing Row order
j+1 to j+2		For each degree of freedom two words are stored: (1) initial displacement (2) initial velocity

Created By: PFBULK

Notes:

1. This entity is used in DMA to assemble the ICMATRIX entity for the boundary condition.
2. The j index runs from 2 to 3 * NROW by 3 for each degree of freedom in the model that has a nonzero initial condition.

Entity: **ICMATRIX**

Entity Type: Matrix

Description: Contains the matrix of transient response initial conditions in the d-set for the current boundary condition.

Matrix Form: A variable-sized rectangular matrix having one row for each degree of freedom in the d-set and two columns for each transient response subcase. The first column is the vector of initial displacement and the second is the vector of initial velocity.

Created By: DMA

Notes:

1. This entity will be flushed for each boundary condition having dynamic response disciplines.

Entity: **ID2**

Entity Type: Matrix

Description: An identity matrix required in the MAPOL sequence associated with the nuclear blast response.

Matrix Form: A 2 by 2 identity matrix.

Created By: Module BLASTFIT

Entity: **IFM**

Entity Type: Subscripted Matrix

Description: Intermediate matrix in the reduction of the mass matrix for unrestrained boundary conditions.

Matrix Form: The number of rows is equal to the number of degrees of freedom in the o-set while the number of columns is equal to the number of degrees of freedom in the a-set.

Created By: MAPOL

Notes:

1. If there are no aerodynamics, IFM is computed from MOO times GSUBO plus MOA.
2. If there are aerodynamics, IFM is computed from MOO times GASUBO plus MOA.
3. Since IFM is required in the sensitivity analysis, it is subscripted by boundary condition number.

Entity: **IFR**

Entity Type: Subscripted Matrix

Description: Intermediate matrix in the reduction of the mass matrix.

Matrix Form: The number of rows is equal to the number of degrees of freedom in the l-set while the number of columns is equal to the number of degrees of freedom in the r-set.

Created By: MAPOL

Notes:

1. IFR is computed as MLL times D plus MLR.
2. Since IFS is required in the sensitivity anaysis, it is subscripted by boundary condition number.

Entity: **IHEX1EST**

Entity Type: Relation

Description: Contains the element summary data for the linear isoparametric hexahedron element.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
SIL_i i=1,...8	Integer > 0	Internal grid points identification numbers
$COORD_i$ i=1,...8	Integer > 0	External coordinate system identification number for displacements at SIL_i
X_i i=1,...8	Real	Basic coordinates of SIL_i
Y_i i=1,...8	Real	
Z_i i=1,...8	Real	
MID	Integer > 0	Material identification number
CID	Integer > 0	Coordinate system identification number in which anisotropic material is defined.
NIP	Integer	Number of integration points in each coordinate direction
AR	Real	Maximum aspect ratio of element
ALFA	Real	Maximum angle (degrees) for face normals
BETA	Real	Maximum angle (degrees) for mid-edge points

Created By: Module MAKEST

Notes:

1. This relation is built from the CIHEX, the PIHEX and the basic grid point relations. It contains one tuple for each linear isoparametric hexahedron element in the problem.

Entity: **IHEX2EST**

Entity Type: Relation

Description: Contains the element summary data for the quadratic isoparametric hexahedron element.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
SILS	Ivector (20)	Internal grid point identification numbers
$COORD_i$ i=1,...20	Integer ≥ 0	External coordinate system identification number for displacements at $SILS_i$
X_i i=1,...20	Rvector (3)	Basic coordinates of $SILS_i$
MID	Integer > 0	Material identification number
CID	Integer > 0	Coordinate system identification number in which anisotropic material is defined
NIP	Integer = 2,3,4	Number of integration points in each coordinate direction
AR	Real	Maximum aspect ratio of element
ALFA	Real	Maximum angle (degrees) for face normals
BETA	Real	Maximum angle (degrees) for mid-edge points

Created By: Module MAKEST

Notes:

1. This relation is built from the CIHEX, the PIHEX, and the basic grid point relations. It contains one tuple for each quadratic isoparametric hexahedron element in the problem.

Entity: **IHEX3EST**

Entity Type: Relation

Description: Contains the element summary data for the cubic isoparametric hexahedron element.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
SILS	Ivector (32)	Internal grid point identification numbers
$COORD_1$ i=1,...32	Integer ≥ 0	External coordinate system identification number for displacements at $SILS_i$
X_i i=1,...32	Rvector (3)	Basic coordinates of $SILS_i$
MID	Integer > 0	Material identification number
CID	Integer > 0	Coordinate system identification number in which anisotropic material is defined
NIP	Integer	Number of integration points in each coordinate direction
AR	Real	Maximum aspect ratio of element
ALFA	Real	Maximum angle (degrees) for face normals
BETA	Real	Maximum angle (degrees) for mid-edge points

Created By: Module MAKEST

Notes:

1. This relation is built from the CIHEX, the PIHEX, and the basic grid point relations. It contains one tuple for each quadratic isoparametric hexahedron element.

Entity: **ITERLIST**

Entity Type: Relation

Description: Contains the definition of the list of design iterations as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer	Set identification number
NITER	Integer	Design iteration number

Created By: Module IFP

Entity: **JOB**

Entity Type: Relation

Description: Contains the case-independent solution control parameters as input in the solution control packet.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
AIRFPRNT	Integer vector (20)	Airfoil shape print selection WORD 1 Print set identification number > 0, or = 0 NONE = –1 ALL = –2 LAST WORD 2 Punch set identification number WORD 3 Print form = 0 Rectangular = 1 Polar WORD 4 Punch form WORD 5 Print frequency set identification number WORD 6 Punch frequency set identification number WORD 7 Print iteration set identification number WORD 8 Punch iteration set identification number WORD 9 Print mode set identification number WORD 10 Punch maode set identification number WORD 11 Print time set identification number WORD 12 Punch time set identification number WORD 13-20 unused
PLANPRNT	Integer vector (20)	Planform print selection
PRESPRNT	Integer vector (20)	Unit pressure coefficient print selection
TITLE	Text (72)	User label TITLE
SUBTITLE	Text (72)	User label SUBTITLE
LABEL	Text (72)	User label LABEL

Created By: Module Solution

Notes:

1. The format of the AIRFPRNT vector is typical of the format of all the print selection vectors. Additionally, the format for the print set Identification number in the AIRFPRNT vector is typical of that of the other set Identification numbers in the vector.
2. The CASE, JOB and OPTIMIZE relation entities together contain the solution control requests as input in the solution control packet. CASE contains the case-dependent parameters, JOB contains the case-independent requests and OPTIMIZE contains the optimization-dependent requests.

Entity: **JSET**

Entity Type: Relation

Description: Contains the external grid identification numbers and components with the analysis set as defined on the JSET entries of the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	JSET identification number
GRID1	Integer > 0	Grid or Scalar point id
COMPNTS	Integer > 0	Component number: = 0 for scalar points = 1-6 for grid points

Created By: Module IFP

Notes:

1. Used by the MKUSET Module to build the USET relation.

Entity: **JSET1**

Entity: Relation

Description: Contains the external grid identification numbers and components associated with the analysis set as defined on the JSET1 entries of the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	JSET identification number
COMPNTS	Integer > 0	Component number: = 0 for scalar points = 1-6 for grid points
GRID1	Integer > 0	Grid or Scalar point id

Created By: Module IFP

Notes:

1. Used by the **MKUSET** Module to build the **USET** relation.

Entity: **KAA**

Entity Type: Matrix

Description: Partition of the KFF matrix (see KGG).

Entity: **KAAA**

Entity Type: Matrix

Description: Partition of the KAFF matrix (see KAFF).

Entity: **KAFF**

Entity Type: Matrix

Description: Contains the stiffness matrix for the free degrees of freedom in the current boundary condition including the aeroelastic terms.

Matrix Form: A variable-sized asymmetric matrix having one row and one column for each free degree of freedom in the current boundary condition.

Created By: MAPOL

Notes:

1. The matrix is formed using

$$[\ \mathbf{KAFF}\ ] = [\ \mathbf{KAFF}\ ] - \bar{q}\,[\ \mathbf{AICS}\ ]$$

2. The MAPOL sequence supports the following partitions of the KAFF matrix (see the Theoretical Manual for the explicit formation of these submatrices):

$$\mathbf{KAFF} \rightarrow \begin{bmatrix} \varphi & \varphi \\ \mathbf{KAO} & \mathbf{KAAA} \end{bmatrix}$$

$$\mathbf{KAAA} \rightarrow \begin{bmatrix} \mathbf{KARR} & \mathbf{KARL} \\ \mathbf{KALR} & \mathbf{KALL} \end{bmatrix}$$

Entity: **KALL**

Entity Type: Matrix

Description: Partition of the KAAA matrix (see KAFF).

Entity: **KALR**

Entity Type: Matrix

Description: Partition of the KAAA matrix (see KAFF).

Entity: **KAO**

Entity Type: Matrix

Description: Partition of the KAFF matrix (see KAFF).

Entity: **KARL**

Entity Type: Matrix

Description: Partition of the KAAA matrix (see KAFF).

Entity:	**KARR**
Entity Type:	Matrix
Description:	Partition of the KAAA matrix (see KAFF).

Entity:	**KDDF**
Entity Type:	Matrix
Description:	Stiffness matrix in the direct set used in frequency response analysis.
Matrix Form:	Complex square matrix with the number of rows and columns equal to the number of degrees of freedom in the d-set.
Created By:	DMA

Entity:	**KDDT**
Entity Type:	Matrix
Description:	Stiffness matrix in the direct set used in frequency response analysis.
Matrix Form:	Complex square matrix with the number of rows and columns equal to the number of degrees of freedom in the d-set.
Created By:	DMA

Entity:	**KEE**
Entity Type:	Matrix
Description:	A partition of the GENK matrix that contains only elastic degrees of freedom. Used in the nuclear blast response analysis.
Matrix Form:	Complex square matrix with the number of rows and columns equal to the number of elastic modes retained for the blast analysis.
Created By:	MAPOL

Entity: **KELM**

Entity Type: Unstructured

Description: Contains the matrix partitions of the element nondesign stiffness and linear design stiffness.

Entity Structure:

Record:

i. If the element is a scalar element:
the record contains the components of the connected grid point(s) (if any) and the value K.

KCODE	FORMAT OF RECORD
1	K
2	COMP1, 0, +–K, –+K
3	0, COMP1, +–K, –+K
4	+–K, –+K
5	COMP1, 0, K
6	COMP1, COMP2, +–K, –+K

Else
the record contains a partition of the stiffness matrix with either 1, 3, or 6 entries for each node

KCODE	FORMAT OF RECORD
7	3 columns of 3 entries/node
8	3 columns of 3 entries/node
9	6 columns of 6 entries/node

Created By: Module EMG

Notes:

1. This entity contains one record for each *strip* of each element stiffness matrix. A strip or partition is defined as all those columns of the element matrix associated with a pivot sil (the id of the first DOF of a grid point or the id of a scalar point).
2. Refer to the DVCT relation for further details, as these two data base entities are closely linked.
3. The matrix partitions are stored in the same precision as the KGG matrix.
4. The records related to nonlinear design stiffness are empty.
5. KELM and DKELM are used to generate all of the element stiffness sensitivity matrix partitions.
6. KELM and KELMD are used to generate all of the element stiffness sensitivity matrix partitions.

Entity: **KELMD**

Entity Type: Unstructured

Description: Contains the element nonlinear design stiffness matrix partitions.

Entity Structure:

Record: the record contains a partition of the stiffness matrix with either 1, 3, or 6 entries for each node

KCODE	FORMAT OF RECORD
7	3 columns of 3 entries/node
8	3 columns of 3 entries/node
9	6 columns of 6 entries/node

Created By: Module EMG

Notes:

1. This entity contains one record for each *strip* of each element nonlinear design stiffness matrix. A strip or partition is defined as all those columns of the element matrix associated with a pivot sil (the id of the first DOF of a grid point or the id of a scalar point).
2. Refer to the DVCTD relation for further details, as these two data base entities are closely linked.
3. The matrix partitions are stored in the same precision as the KGG matrix.
4. KELMD and KELM are used to generate all of the element stiffness sensitivity matrix partitions.

Entity: **KEQE**

Entity Type: Matrix

Description: Structural plus aerodynamic stiffness matrix for the elastic modes retained in the blast response analysis.

Matrix Form: Complex square matrix with the number of rows and columns equal to the number of elastic modes retained for the blast analysis.

Created By: MAPOL

Notes:

1. KEQE is obtained from **[KEQE] = [KEE] + [QEE]**

Entity: **KFF**

Entity Type: Matrix

Description: Partition of the KNN matrix (see KGG).

Entity: **KFS**

Entity Type: Matrix

Description: Partition of the KNN matrix (see KGG).

Entity: **KGG**

Entity Type: Matrix

Description: Contains the current global stiffness matrix for the design problem.

Matrix Form: A variable-size symmetric matrix having one row and one column for each structural degree of freedom in the problem.

Created By: Module MAPOL

Notes:

1. The KGG matrix is formed in the second phase stiffness matrix assembly.
2. The MAPOL sequence supports the following partitions of the KGG matrix (see the Theoretical Manual for the explicit formation of these submatrices):

$$\mathbf{KGG} \rightarrow \begin{bmatrix} \varphi & \varphi \\ \varphi & \mathbf{KNN} \end{bmatrix}$$

$$\mathbf{KNN} \rightarrow \begin{bmatrix} \mathbf{KSS} & \varphi \\ \mathbf{KFS} & \mathbf{KNN} \end{bmatrix}$$

$$\mathbf{KFF} \rightarrow \begin{bmatrix} \mathbf{KOO}* & \mathbf{KOA}* \\ \varphi & \mathbf{KAA} \end{bmatrix}$$

$$\mathbf{KAA} \rightarrow \begin{bmatrix} \varphi & \varphi \\ \mathbf{KLR} & \mathbf{KLL} \end{bmatrix}$$

* Generalized dynamic reduction only

Entity: **KHHF**

Entity Type: Subscripted Matrix

Description: Stiffness matrix in the modal set used in frequency response and flutter analyses.

Matrix Form: Complex square matrix with the number of rows and columns equal to the number of degrees of freedom in the h-set.

Created By: DMA

Notes:

1. The matrix may be required in the flutter sensitivity analysis and is therefore subscripted by boundary condition.

Entity:	**KHHT**
Entity Type:	Subscripted Matrix
Description:	Stiffness matrix in the modal set used in transient response analysis.
Matrix Form:	Complex square matrix with the number of rows and columns equal to the number of degrees of freedom in the h-set.
Created By:	DMA

Entity:	**KLL**
Entity Type:	Matrix
Description:	Partition of the KAA matrix (see KGG).

Entity:	**KLLINV**
Entity Type:	Subscripted Matrix
Description:	Contains the upper and lower triangular sections of the [KLL] symmetric stiffness matrix.
Matrix Form:	A variable-sized matrix having one row and one column for each degree of freedom left over for analysis after partition/reduction. The precision of this matrix is the same as the KGG matrix.
Created By:	SDCOMP
Notes:	1. This matrix is formed for use by the FBS utility.

Entity:	**KLLL**
Entity Type:	Matrix
Description:	Contains the lower triangular portion of the decomposed KAAA matrix. Note that KAAA is asymmetric requiring use of the general decomposition routine.
Matrix Form:	Refer to the DECOMP utility documentation.
Created By:	DECOMP
Notes:	1. This matrix is formed to be used in the general forward backward substitution module GFBS.

Entity:	**KLLU**
Entity Type:	Matrix
Description:	Contains the upper triangular portion of the decomposed partition of the KAAA matrix. Note that KAAA is asymmetric requiring use of the general decomposition routine.
Matrix Form:	Refer to the DECOMP utility documentation.
Created By:	DECOMP
Notes:	1. This matrix is formed to be used in the general forward backward substitution module GFBS.

Entity:	**KLR**
Entity Type:	Matrix
Description:	A partition of the KAA matrix (see KGG).

Entity:	**KL11**
Entity Type:	Subscripted Matrix
Description:	Lower triangular portion of the decomposed K11 matrix.
Matrix Form:	Square real matrix having one row and one column for each a-set degree of freedom.
Created By:	DECOMP
Notes:	1. K11 is not symmetric. 2. This matrix is formed for use by the GFBS utility. 3. The matrix may be required in the sensitivity analysis and is therefore subscripted by boundary condition.

Entity:	**KNN**
Entity Type:	Matrix
Description:	A partition of the KGG matrix (see KGG).

Entity:	**KOA**
Entity Type:	Matrix
Description:	A partition of the KFF matrix used in Generalized Dynamic reduction.
Matrix Form:	Real rectangular matrix with one row for each o-set degree of freedom and one column for each a-set degree of freedom.
Created By:	PARTN
Notes:	1. This matrix is required only for generalized dynamic reduction and only when the user has specified a-set degrees of freedom.

Entity:	**KOO**
Entity Type:	Matrix
Description:	A partition of the KFF matrix used in Generalized Dynamic reduction.
Matrix Form:	Real squar matrix with one row for each o-set degree of freedom.
Created By:	PARTN
Notes:	1. This matrix is required only for generalized dynamic reduction. 2. If the user has not specified a-set degrees of freedom, KOO is equivalented to KFF.

Entity:	**KOOINV**
Entity Type:	Subscripted Matrix
Description:	Contains the upper and lower triangular matrices resulting from the symmetric decomposition of the KOO matrix for the current boundary condition.
Matrix Form:	A variable-sized matrix having one row and one column for each omitted degree of freedom in the boundary condition.
Created By:	FREDUCE
Notes:	1. This matrix has the same precision as the global stiffness matrix. 2. If no degrees of freedom have been omitted in the current boundary condition, this matrix will have no rows or columns. 3. This matrix is formed for use by the Forward Backward Substitution Utility. 4. Note that KOOINV plays the same role as KOOL/KOOU for symmetric analyses.

Entity: **KOOL**

Entity Type: Subscripted Matrix

Description: Contains the lower triangular portion of the decomposed partition of the KAAA matrix. Note that KAAA is asymmetric requiring use of the general decomposition routine.

Matrix Form: Refer to the DECOMP utility documentation.

Created By: FREDUCE

Notes:

1. This matrix is formed to be used in the general forward backward substitution module GFBS.
2. Note that KOOL/KOOU play the same role as KOOINV for asymmetric analyses.

Entity: **KOOU**

Entity Type: Subscripted Matrix

Description: Contains the upper triangular portion of the decomposed partition of the KAAA matrix. Note that KAAA is asymmetric requiring use of the general decomposition routine.

Matrix Form: Refer to the DECOMP utility documentation.

Created By: Module DECOMP

Notes:

1. This matrix is formed to be used in the general forward backward substitution module GFBS.
2. Note that KOOL/KOOU play the same role as KOOINV for asymmetric analyses.

Entity: **KSOO**

Entity Type: Matrix

Description: Shifted stiffness matrix used in Generalized Dynamic reduction.

Matrix Form: Real square symmetric matrix with one row and one column for each o-set of degree of freedom.

Created By: Module GDR1

Notes:

1. This matrix is computed from: `[KSOO] = [KOO] -s[MOO];` with the shift parameter, s, computed in GDR1.

Entity: **KSS**

Entity Type: Matrix

Description: Partition of the KNN matrix (see KGG).

Entity: **KU11**

Entity Type: Subscripted Matrix

Description: Upper triangular portion of the decomposed K11 matrix.

Matrix Form: Square real matrix having one row and one column for each a-set degree of freedom.

Created By: DECOMP

Notes:

1. K11 is not symmetric.
2. This matrix is formed for use by the GFBS utility.
3. The matrix may be required in the sensitivity analysis and is therefore subscripted by boundary condition.

Entity: **K11**

Entity Type: Matrix

Description: An intermediate matrix that is constructed as part of the solution of unrestrained structures.

Matrix Form: A real, square matrix with the number of rows and columns equal to the number of a-set degrees of freedom.

Created By: MAPOL

Entity: **K1112**

Entity Type: Subscripted Matrix

Description: An intermediate matrix required in the static aeroelastic trim analysis.

Matrix Form: A real rectangular matrix with the number of rows equal to the number of a-set degrees of freedom and the number of columns equal to the number of r-set degrees of freedom.

Created By: MAPOL

Notes:

1. The K1112 matrix is computed as the solution of

   [K11][K1112] = [AICS]

2. The matrix may be required in the sensitivity analysis and is therefore subscripted by boundary condition.

Entity: **K12**

Entity Type: Subscripted Matrix

Description: An intermediate matrix that is constructed as part of the solution of unrestrained structures.

Matrix Form: A real rectangular matrix with the number of rows equal to the number of a-set degrees of freedom and the number of columns equal to the number of r-set degrees of freedom.

Created By: MAPOL

Notes:

1. The matrix may be required in the sensitivity analysis and is therefore subscripted by boundary condition.

Entity: **K21**

Entity Type: Subscripted Matrix

Description: An intermediate matrix used in the reduction of the aerodynamic stiffness.

Matrix Form: The number of rows is equal to the number of degrees of freedom in the r-set and the number of columns is equal to the number of rows in the a-set.

Created By: MAPOL

Notes:

1. The matrix is created from the column merge of R32 and R31.
2. Since this matrix is required in the sensitivity analysis, it is subscripted by the boundary condition number.

Entity: **LAMBDA**

Entity Type: Relation

Description: Contains the results of real eigenvalue analysis for each modal analysis in each boundary condition.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
NITER	Integer	Iteration number
BCID	Integer	The boundary condition number
MODENO	Integer	The mode number of the eigenvalue/eigenvector
EXORD	Integer	The extraction order for the mode
EIGVAL	Real	The eigenvalue
RFREQ	Real	The modal frequency in rad/s
CFREQ	Real	The modal frequency in Hertz
VECFLG	Integer	= 1 if a vector was generated for the mode = 0 if only the value was extracted
GMASS	Real	The generalized mass associated with the mode
GSTIFF	Real	The generalized stiffness associated with the mode

Created By: Module REIG

Notes:

1. The relation contains one tuple for each mode extracted in each eigenanalysis.
2. All eigenvalues for all boundary conditions at each design iteration are stored for retrieval in sensitivity evaluation.

Entity: **LAMDAC**

Entity Type: Relation

Description: Contains the results of complex eigenvalue analysis for each modal analysis in each boundary condition.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
BCID	Integer	The boundary condition number
MODENO	Integer	The mode number of the eigenvalue/eigenvector
EXORD	Integer	The extraction order for the mode
REIGVAL	Real	The real part of the eigenvalue
IEIGVAL	Real	The imaginary part of the eigenvalue
NFREQ	Real	The natural frequency in Hertz
DFREQ	Real	The damped frequency in Hertz
DPCOEF	Real	The damping coefficient

Created By: Module CEIG

Notes:

1. The relation contains one tuple for each mode extracted in each eigenanalysis.

Entity: **LDVLIST**

Entity Type: Relation

Description: Contains the definition of the list of local design iterations as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer	Set identification number
ETYPE	Text(8)	Element type
LAYRNUM	Integer	Layer number for composites or zero
DVSYMBL	Text(8)	Designed dimension symbol (D1, D2, D3, D4, D5, D6, D7, D8, D9, D10 for BAR element cross section dimension; A for element area, T for element thickness; M for element mass; K for element stiffness.)
EID	Integer	Element identification number

Created By: Module IFP

Entity: **LHS**

Entity Type: Subscripted Matrix

Description: This is essentially a matrix of rigid body mass values with the exact definition depending on the type of free-free analysis being performed.

Matrix Form: The dimension of the square matrix is equal to the number of degrees of freedom in the r-set.

Created By: MAPOL

Notes:

1. For an inertia relief analysis, LHS is equal to MRR.
2. For a static aeroelastic analysis, LHS is equal to MRR plus K21 times K1112.

Entity: **LKQ**

Entity Type: Matrix

Description: Lower triangular portion of the decomposed KEQE matrix.

Matrix Form: Square real matrix having one row and one column for each elastic mode retained in the nuclear blast response analysis.

Created By: DECOMP

Notes:

1. KEQE is not symmetric.
2. This matrix is formed for use by the GFBS utility.

Entity: **LOAD**

Entity Type: Relation

Description: Contains the definition of a static load that is a linear combination of loads defined by FORCE, MOMENT, FORCE1, MOMENT1, PLOAD, and GRAV entries.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
SCALE	Real	Scale factor for combination
SCALEI	Real	Scale factor for component load
LOADI	Integer > 0	Set identification number of the component load

Created By: Module IFP

Notes:

1. The relation contains one tuple for each load set id specified in each unique SETID.

Entity: **LOCLVAR**

Entity Type: Relation

Description: Contains the relationship between local variables and global variables in the design problem. Acts as a pointer to the PTRANS matrix.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification
ETYPE1	Text (8)	Element type
PID	Integer > 0	Element property identification
PTYPE	Text (8)	Element property type
LAYRNUM	Integer ≥ 0	Layer number for composites
DVSYMBL	Text(8)	Designed dimension symbol (D1, D2, D3, D4, D5, D6, D7, D8, D9, D10 for BAR element cross section dimension; A for element area, T for element thickness; M for element mass; K for element stiffness.)
PROW	Integer > 0	Pointer to (P) row for this element
PTYP	Integer = 1, 2 or 3	Flag indicating type of associated global variable
TMIN	Real	Minimum value for physical property
TMAX	Real	Maximum value for physical property

Created By: Module MAKEST

Notes:

1. This entity is used to create move limits on the physical design variables in the TCEVAL module.
2. The PTYP attribute indicates the linking option for the physical variable
   = 1 unique physical linking (DESELM)
   = 2 physical linking (DESVARP)
   = 3 shape function linking (DESVARS)

Entity: **LSOO**

Entity Type: Matrix

Description: Lower triangular portion of the decomposed KSOO matrix.

Matrix Form: Square real matrix having one row and column for each o-set degree of freedom in Generalized Dynamic Reduction.

Created By: DECOMP

Notes:

1. This matrix is formed for use by the FBS large matrix utility.
2. LSOO is computed only when there are k-set degrees of freedom in a generalized dynamic reduction analysis.

Entity: **MAA**

Entity Type: Matrix

Description: Mass matrix in the a-set derived from partitions of the MFF matrix (see MGG).

Entity: **MAABAR**

Entity Type: Matrix

Description: A partition of the MFF matrix (see MGG).

Entity: **MACHLIST**

Entity Type: Relation

Description: Contains the list of Mach numbers as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer	Set identification number
MACH	Real > 0.0	Mach number

Created By: Module IFP.

Entity: **MASDRVA**

Entity Type: Relation

Description: Contains the user function requested element mass sensitivity information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
MXCOL	Integer > 0	Matrix column number in [MXMASDVA] for the sensitivity values

Created By: Module MSWGGRAD

Entity: **MASRESP**

Entity Type: Relation

Description: Contains the user function requested element mass values.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
VALUE	Real	Response value

Created By: Module MSWGRESP

Entity: **MASSEST**

Entity Type: Relation

Description: Contains the element summary data for the MASS1 and MASS2 elements.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
SIL1	Integer ≥ 0	Internal grid or scalar point id
SIL2	Integer ≥ 0	Internal grid or scalar point id
COMPNT1	Integer ≥ 0	Component of SIL1 to which the element is attached
COMPNT2	Integer ≥ 0	Component of SIL2 to which the element is attached
MASS	Real	Mass value
DESIGN	Integer > 0	Design flag

Created By: Module MAKEST

Notes:

1. A nonzero design flag denotes that the element is affected by a design variable.
2. This relation is built from the CMASS1 and CMASS2 relations along with associated property, design and grid relations. It contains one tuple for each scalar mass element in the problem.

Entity: **MATSS**

Entity Type: Matrix

Description: Matrix of steady-state influence coefficients used in the nuclear blast response analysis.

Matrix Form: Real square matrix with one row and column for each panel in the unsteady aerodynamics model.

Created By: Module BLASTFIT

Entity: **MATTR**

Entity Type: Matrix

Description: Matrix of transient influence coefficients used in the nuclear blast response analysis.

Matrix Form: Real rectangular matrix with one row for each panel in the unsteady aerodynamics model and a column dimension equal to the number of panels times the number of beta values used in fitting procedure of the transient blast response calculations.

Created By: Module BLASTFIT

Entity: **MAT1**

Entity Type: Relation

Description: Contains the material properties for linear isotropic materials as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
MID	Integer > 0, key	Material property identification
E	Real	Young's Modulus
G	Real	Shear Modulus
NU	Real	Poisson's Ratio
RHO	Real	Density
ALPHA	Real	Thermal expansion coefficient
TREF	Real	Thermal expansion reference temperature
DAMPING	Real	Structural damping coefficient
ST	Real	Tension stress allowable
SC	Real	Compression stress allowable
SS	Real	Shear stress allowable
MSCID	Integer ≥ 0	Material coordinate system id

Created By: Module IFP

Entity: **MAT2**

Entity Type: Relation

Description: Contains the material properties for linear anisotropic materials for two-dimensional elements as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
MID	Integer > 0, key	Material identification number
G11, G12, G13	Real	Components of the 6 x 6 symettric material properties matrix
G22, G23, G33	Real	
RHO	Real	Density
ALPH1	Real	Thermal expansion coefficient vector
ALPH2	Real	
ALPH12	Real	
TREF	Real	Thermal expansion reference temperature
DAMPING	Real	Structural damping coefficient
ST	Real	Tension stress allowable
SC	Real	Compression stress allowable
SS	Real	Shear stress allowable
MSCID	Integer ≥ 0	Material coordinate system id

Created By: Module IFP

Entity: **MAT8**

Entity Type: Relation

Description: Contains the material properties for orthotropic materials for two-dimensional elements as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
MID	Integer > 0, key	Material identification number
E1	Real ≠ 0.0	Logitudinal modulus of elasticity
E2	Real ≠ 0.0	Transverse modulus of elasticity
NU12	Real	Poisson's ratio
G12	Real > 0.0	In-plane shear modulus
G1Z	Real ≥ 0.0	Transverse shear modulus in 1-z plane
G2Z	Real ≥ 0.0	Transverse shear modulus in 2-z plane
RHO	Real ≥ 0.0	Mass density
ALPH1	Real	Thermal expansion coefficient in 1-direction
ALPH2	Real	Thermal expansion coefficient in 2-direction
TREF	Real	Element reference temperature
XT	Real ≥ 0.0	Allowable longitudinal tension stress
XC	Real	Allowable longitudinal compression stress
YT	Real ≥ 0.0	Allowable transverse tension stress
YC	Real	Allowable transverse compression stress
SS	Real ≥ 0.0	Allowable in-plane shear stress
DAMPING	Real	Structural damping value
F12	Real	Tsai Wu tensor polynominal theory interaction term

Created By: Module IFP

Entity: **MAT9**

Entity Type: Relation

Description: Contains the material properties for orthotropic materials for three-dimensional elements as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
MID	Integer > 0, key	Material identification number
G11	Real	Tensile modulus in the 1-direction
G12	Real	Shear modulus in the 1-2 plane
G13	Real	Shear modulus in the 1-3 plane
G14	Real	Shear modulus in the 1-4 plane
G15	Real	Shear modulus in the 1-5 plane
G16	Real	Shear modulus in the 1-6 plane
G22	Real	Tensile modulus in the 2-direction
G23	Real	Shear modulus in the 2-3 plane
G24	Real	Shear modulus in the 2-4 plane
G25	Real	Shear modulus in the 2-5 plane
G26	Real	Shear modulus in the 2-6 plane
G33	Real	Tensile modulus in the 3-direction
G34	Real	Shear modulus in the 3-4 plane
G35	Real	Shear modulus in the 3-5 plane
G36	Real	Shear modulus in the 3-6 plane
G44	Real	Tensile modulus in the 4-direction
G45	Real	Shear modulus in the 4-5 plane
G46	Real	Shear modulus in the 4-6 plane
G55	Real	Tensile modulus in the 5-direction
G56	Real	Shear modulus in the 5-6 plane
G66	Real	Tensile modulus in the 6-direction
RHO	Real	Mass Density
ALPH1	Real	Thermal expansion coefficient in 1-direction
ALPH2	Real	Thermal expansion coefficient in 2-direction
ALPH3	Real	Thermal expansion coefficient in 3-direction
ALPH4	Real	Thermal expansion coefficient in 4-direction
ALPH5	Real	Thermal expansion coefficient in 5-direction

NAME	TYPE/KEY	DESCRIPTION
ALPH6	Real	Thermal expansion coefficient in 6-direction
TREF	Real	Element reference temperature
GE	Real	Structural damping coefficient

Created By: Module IFP

Entity: **MDD**

Entity Type: Matrix

Description: Mass matrix in the direct set.

Matrix Form: Square matrix with the number of rows and columns equal to the number of degrees of freedom in the d-set.

Created By: DMA

Entity: **MELM**

Entity Type: Unstructured

Description: An unstructured database entity that contains the matrix partitions of nondesigned mass and linearly designed mass.

Entity Structure:

Record:

i. If the element is a scalar element:
the record contains the components of the connected grid point(s) (if any) and the value M.

MCODE	FORMAT OF RECORD
1	M
2	COMP1, 0, +–M,–+M
3	0,COMP1, +–M,–+M
4	+–M,–+M
5	COMP1, 0, M
6	COMP1, COMP2, +–M,–+M

Else
the record contains a partition of the stiffness matrix with either one, three, or six entries for each node

KCODE	FORMAT OF RECORD
7	3 columns of 3 entries/node
10	3 columns of 1 entry/node (diagonal)

Created By: Module EMG

Notes:

1. This entity contains one record for each *strip* of each element mass matrix. A strip or partition is defined as all those columns of the element matrix associated with a pivot sil (the id of the first dof of a grid point or the id of a scalar point).
2. Refer to the DVCT relation for further details, as these two database entities are closely linked.
3. The matrix partitions are stored in the same precision as the MGG matrix.
4. The records related to nonlinear design mass are empty.
5. MELM and DMELM are used to generate all of the element mass sensitivity matrix partitions.
6. MELM and MELMD are used to generate all of the element mass sensitivity matrix partitions.

Entity: **MELMD**

Entity Type: Unstructured

Description: An unstructured database entity that contains the element nonlinear design mass matrix partitions.

Entity Structure:

Record: the record contains a partition of the stiffness matrix with either one, three, or six entries for each node

KCODE	FORMAT OF RECORD
7	3 columns of 3 entries/node
10	3 columns of 1 entry/node (diagonal)

Created By: Module EMG

Notes:

1. This entity contains one record for each *strip* of each element nonlinear design mass matrix. A strip or partition is defined as all those columns of the element matrix associated with a pivot sil (the id of the first dof of a grid point or the id of a scalar point).
2. Refer to the DVCTD relation for further details, as these two database entities are closely linked.
3. The matrix partitions are stored in the same precision as the MGG matrix.
4. MELMD and MELM are used to generate all of the element mass sensitivity matrix partitions.

Entity: **MFF**

Entity Type: Matrix

Description: A partition of the MNN matrix (see MGG).

Entity: **MFORM**

Entity Type: Relation

Description: Contains the mass matrix form as specified in the Bulk Data file.

Relation Attributes:

NAMES	TYPE/KEY	DESCRIPTION
VALUE	Text (8)	The mass matrix form; either LUMPED or COUPLED.

Notes:

1. If this relation is empty, the LUMPED form will be used. If more than one tuple is defined, any tuple containing the "COUPLED" option will cause the coupled mass form to be used.

Entity: **MGG**

Entity Type: Matrix

Description: Contains the current global mass matrix for the design problem.

Matrix Form: A variable-size symmetric (possibly diagonal) matrix having one row and one column for each structural degree of freedom in the problem.

Created By: Module MAPOL

Notes:

1. The MGG matrix is formed in the second phase mass matrix assembly.
2. The MAPOL sequence supports the following partitions of the MGG matrix (see Theoretical Manual for the explicit formation of these matrices):

$$\mathbf{MGG} \rightarrow \begin{bmatrix} \varphi & \varphi \\ \varphi & \mathbf{MNN} \end{bmatrix} ; \quad \mathbf{MNN} \rightarrow \begin{bmatrix} \varphi & \varphi \\ \varphi & \mathbf{MFF} \end{bmatrix}$$

$$\mathbf{MFF} \rightarrow \begin{bmatrix} \mathbf{MOO} & \mathbf{MOA} \\ \varphi & \mathbf{MAABAR} \end{bmatrix}$$

$$\mathbf{MAA} \rightarrow \begin{bmatrix} \mathbf{MRRBAR} & \varphi \\ \mathbf{MLR} & \mathbf{MLL} \end{bmatrix}$$

Entity: **MHH**

Entity Type: Subscripted Matrix

Description: Contains the modal mass output from the dynamic matrix assembly.

Matrix Form: A variable-sized matrix having one row and one column for each eigenvector computed in the real eigenanalysis.

Created By: Module DMA

Notes:

1. This matrix is needed for flutter constraint sensitivities so it is subscripted for each boundary condition.
2. INFO(11) contains a flag denoting whether the matrix is coupled or uncoupled
   = 0 Uncoupled
   = 1 Coupled

Entity: **MII**

Entity Type: Matrix

Description: Generalized mass matrix computed by the eigenanalysis module.

Matrix Form: Square diagonal matrix with the number of rows and columns equal to the number of modes retained by the eigenanalysis.

Created By: Module REIG

Notes:

1. Currently, this matrix is computed and not used; it is available for printout.

Entity: **MKAERO1**

Entity Type: Relation

Description: Contains a table of Mach numbers and reduced frequencies for unsteady aerodynamic matrix calculation as input from the bulk data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SYMXZ	Integer	Symmetry flag for xz-plane
SYMXY	Integer	Symmetry flag for xy-plane
MACH1	Real	Mach number
MACH2	Real	
MACH3	Real	
MACH4	Real	
MACH5	Real	
MACH6	Real	
RFREQ1	Real	Reduced frequencies
RFREQ2	Real	
RFREQ3	Real	
RFREQ4	Real	
RFREQ5	Real	
RFREQ6	Real	
RFREQ7	Real	
RFREQ8	Real	

Created By: Module IFP

Entity: **MKAERO2**

Entity Type: Relation

Description: Contains mach number and reduced frequency pairs to be used in unsteady aerodynamic matrix generation.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SYMXZ	Integer	Symmetry flag for xz-plane
SYMXY	Integer	Symmetry flag for xy-plane
MACH	Real ≥ 0.0	Mach number
RFREQ	Real ≥ 0.0	Reduced frequency

Created By: Module IFP

Entity: **MLL**

Entity Type: Matrix

Description: A partition of the MAA matrix (see MGG).

Entity: **MLR**

Entity Type: Matrix

Description: A partition of the MAA matrix (see MGG).

Entity: **MNN**

Entity Type: Matrix

Description: The mass matrix in the n-set derived from partition of the MGG matrix (see MGG).

Entity: **MOA**

Entity Type: Matrix

Description: A partition of the MFF matrix (see MGG).

Entity: **MODELIST**

Entity Type: Relation

Description: Contains the list of normal modes for which outputs are requested as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer	Set identification number
MODE	Integer	Mode number

Created By: Module IFP

Entity: **MOMENT**

Entity Type: Relation

Description: Contains the definition of a static moment at a grid point as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
GRID1	Integer > 0	Grid point at which the moment is applied
CID1	Integer ≥ 0	Coordinate system identification
SCALE	Real	Scale factor
N1, N2, N3	Real	Components of the vector

Created By: Module IFP

Entity: **MOMENT1**

Entity Type: Relation

Description: Contains the definition of a moment applied at a grid point with the direction determined by a line connecting two grid points.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
GRID1	Integer > 0	Grid point id at which the moment is applied
SCALE	Real	Scale factor
GRID2	Integer > 0	Grid point identification
GRID3	Integer > 0	Grid point identification

Created By: Module IFP

Entity: **MOO**

Entity Type: Matrix

Description: A partition of the MFF matrix (see MGG).

Entity: **MPART**

Entity Type: Matrix

Description: A partitioning vector used to separate rigid body and elastic modes for nuclear blast response.

Matrix Form: A real vector with the number of rows equal to the number of modes retained for the nuclear blast response. Rigid modes are denoted by 0.0 while elastic modes are denoted by 1.0.

Created By: Module BLASTFIT

Entity: **MPC**

Entity Type: Relation

Description: Contains the multipoint constraint data as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
DEPEND	Integer > 0	Dependent grid or scalar point id
COMPNT1	Integer ≥ 0	Component of DEPEND that is constrained
DEPCOEF	Real	Coefficient of constraint for the dependent dof
GRID2	Integer > 0	Grid or scalar point id
COMPNT2	Integer > 0	Component of GRID2 that specifies a constraint
MPCCOEF	Real	Coefficient of constraint

Created By: Module IFP

Notes:

1. The relation contains one tuple for each component constrained in each unique SETID.

Entity: **MPCADD**

Entity Type: Relation

Description: Contains the definition of a multipoint constraint set that is a union of sets contained in the MPC relation.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
MPCSETID	Integer > 0	The SETID of the MPC relation tuples to be used

Created By: Module IFP

Entity: **MPPARM**

Entity Type: Realtion

Description: Contains the optimizer parameters and their new values for use in the ASTROS mathematical programming optimizer as input from the Bulk Data File.

NAME	TYPE/KEY	DESCRIPTION
PARAM	Text (8)	Name of the parameter
INTPARM	Integer	Value of integer parameters
RSPPARM	Real	Value of real parameters

Created By: Module IFP

Notes:

1. This relation is used in module DESIGN to provide for user specification of optimizer parameters.

Entity: **MRR**

Entity Type: Subscripted Matrix

Description: To reduce mass matrix for the structural model.

Matrix Form: A variable-sized matrix having one row and one column for each degree of freedom in the support set for the current boundary condition.

Created By: MAPOL

Notes:

1. This matrix is required to compute strength constraint sensitivities for unrestrained structures and trim parameter sensitivities for steady aerolastic optimization so it is subscripted for each boundary condition.

Entity: **MRRBAR**

Entity Type: Matrix

Description: A partition of the MAA matrix (see MGG).

Entity: **MXDSPDVA**

Entity Type: Matrix

Description: Matrix contains displacement response sensitivity to the design variables.

Matrix Form: A variable-sized matrix having one row for each design variable and one column for each requested displacement response.

Created By: MKAMAT

Entity:	**MXELMDVA**
Entity Type:	Matrix
Description:	Matrix contains element stress and strain response sensitivity to the design variables.
Matrix Form:	A variable-sized matrix having one row for each design variable and one column for each requested element stress/strain response.
Created By:	MKAMAT

Entity:	**MXFCFDVA**
Entity Type:	Matrix
Description:	Matrix contains steady aeroelastic flexible stability coefficient response sensitivity to the design variables.
Matrix Form:	A variable-sized matrix having one row for each design variable and one column for each requested flexible stability coefficient response.
Created By:	AEROEFFS

Entity:	**MXFDPDVA**
Entity Type:	Matrix
Description:	Matrix contains flutter damping response sensitivity to the design variables.
Matrix Form:	A variable-sized matrix having one row for each design variable and one column for each requested flutter damping response.
Created By:	FLUTSENS

Entity:	**MXFFQDVA**
Entity Type:	Matrix
Description:	Matrix contains flutter freqency response sensitivity to the design variables.
Matrix Form:	A variable-sized matrix having one row for each design variable and one column for each requested flutter frequency response.
Created By:	FLUTSENS

Entity:	**MXFRQDVA**
Entity Type:	Matrix
Description:	Matrix contains normal mode response sensitivity to the design variables.
Matrix Form:	A variable-sized matrix having one row for each design variable and one column for each requested normal mode response.
Created By:	FREQSENS

Entity:	**MXFRTDVA**
Entity Type:	Matrix
Description:	Matrix contains flutter root response sensitivity to the design variables.
Matrix Form:	A variable-sized matrix having one row for each design variable and two columns for each requested flutter root response (the first column for the real part and the second column for the imaginary part).
Created By:	MKAMAT

Entity:	**MXGDVDVA**
Entity Type:	Matrix
Description:	Matrix contains global design variable sensitivity to the design variables.
Matrix Form:	A variable-sized matrix having one row for each design variable and one column for each requested global design variable intrinsic.
Created By:	GDVGRAD

Entity:	**MXMASDVA**
Entity Type:	Matrix
Description:	Matrix contains element mass intrinsic sensitivity to the design variables.
Matrix Form:	A variable-sized matrix having one row for each design variable and one column for each requested element mass intrinsic.
Created By:	MSWGGRAD

Entity:	**MXTHKDVA**
Entity Type:	Matrix
Description:	Matrix contains element thickness intrinsic sensitivity to the design variables.
Matrix Form:	A variable-sized matrix having one row for each design variable and one column for each requested element thickness intrinsic.
Created By:	MAKDFV

Entity:	**MXTRMDVA**
Entity Type:	Matrix
Description:	Matrix contains steady aeroelastic trim parameter response sensitivity to the design variables.
Matrix Form:	A variable-sized matrix having one row for each design variable and one column for each requested trim parameter response.
Created By:	AEROSENS

Entity: **MXWGHDVA**

Entity Type: Matrix

Description: Matrix contains element weight intrinsic sensitivity to the design variables.

Matrix Form: A variable-sized matrix having one row for each design variable and one column for each requested element weight intrinsic.

Created By: MSWGGRAD

Entity: **NLGLBSIG**

Entity Type: Matrix

Description: Contains the stress and strain components in the element coordinate system for elements nonlinearly constrained through stress/strain constraint bulk data entries.

Matrix Form: A variable-size matrix having one row for each stress/strain component for each element subject to a nonlinear strength constraint and one column for each load condition within each boundary condition. The order of the matrix rows is in element id order of linearly constrained elements within each element type. The element types are currently processed in the following order:

(1) BAR; $^{\sigma}a1$, $^{\sigma}a2$, $^{\sigma}a3$, $^{\sigma}a4$, $^{\sigma}b1$, $^{\sigma}b2$, $^{\sigma}b3$, $^{\sigma}b4$
(2) QUAD4; $^{\sigma}x$, $^{\sigma}y$, $^{\tau}xy$
(3) TRIA3; $^{\sigma}x$, $^{\sigma}y$, $^{\tau}xy$
The columns are processed in load condition order for each boundary condition.

Created By: Module SCEVAL

Notes:

1. If no elements are nonlinearly constrained, this matrix will be empty.
2. Refer to the NLSMAT documentation for further details as NLGLBSIG is essentially:

   $[\mathbf{NLSMAT}]^{t}\ [u_g]$

3. Each boundary condition's load conditions are appended onto the existing NLGLBSIG columns within the SCEVAL module.

Entity: **NLSMAT**

Entity Type: Matrix

Description: Contains the nonlinear portion of sensitivity of the stress and strain in the elements coordinate system to the global displacements.

Matrix Form: A variable-sized matrix having one column for every stress/strain term in each element that is nonlinearly constrained by a stress/strain constraint tuple or its stress/strain responses are required by user functions, and one row for every structural degree of freedom. The columns are stored in the order the nonlinearly constrained elements are processed in EMG. Relation NLSMTCOL has the matrix [NLSMAT] column information for each related element.

Created By: Module NLEMG

Notes:

1. This matrix is not built if all elements' stress or strain constraints are linear in the design variables.
2. NLSMAT and SMAT are used by SCEVAL module for constraint evaluation and MAKDFU for sensitivity evaluation.

Entity: **NLSMTCOL**

Entity Type: Relation

Description: Contains the matrix [NLSMAT] column information for the realted elements.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0	Element identification number
ETYPE	Text (8)	Element type
LAYERNUM	Integer ≥ 0	Composite layer number
STRECOL	Integer > 0	The first column number in [NLSMAT] for this element/layer stress
NCOLSTRE	Integer > 0	The number of columns in [NLSMAT] for this element/layer stress
STRACOL	Integer > 0	The first column number in [NLSMAT] for this element/layer strain
NCOLSTRA	Integer > 0	The number of columns in [NLSMAT] for this element/layer strain

Created By: Module NLEMG

Entity: **OAGRDDSP**

Entity Type: Relation

Description: Contains the displacements on the aerodynamic boxes ("grids") for static aeroelasticity, flutter, transient/gust and blast disciplines that are requested for print or punch in Solution Control.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
NITER	Integer	Iteration number
BCID	Integer	The boundary condition number
DISC	Integer	Discipline type flag from CASE
SUBCASE	Integer	Subcase identification number from CASE relation
MODENO	Integer	Normal mode number for FLUTTER
CMPLX	Integer	Real or complex flag = 1 if real displacement = 2 if complex displacement
EXTID	Integer	External identification number of the aerodynamic box (See Remark 1)
INTID	Integer	Internal identification number of the aerodynamic box
RDISP	Real	Real part of the normal displacement
IDISP	Real	Imaginary part of the normal displacement

Created By: Many Modules

Notes:

1. The "grids" referred to by the EXTID are actually the aerodynamic box elements. Each of these elements is physically located at the centroid of a quadrilateral or triangular plate (the location of which is stored in GEOMSA or GEOMUA depending on the model).
2. The DISC flag also indicates which model is referred to by the results:
   SAERO refers to the planar static aero model
   FLUTTER, TRANSIENT, FREQUENCY and BLAST refer to the unsteady aero model

Entity: **OAGRDLOD**

Entity Type: Relation

Description: Contains the trimmed applied steady aerodynamic forces and pressures on the planar and nonplanar static aerodynamic boxes ("grids") that are requested for print or punch in Solution Control.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
NITER	Integer	Iteration number
BCID	Integer	The boundary condition number
DISC	Integer	Discipline type flag from CASE
SUBCASE	Integer	Subcase identification number (Normal mode number for FLUTTER)
LOADTYPE	Text(8)	Label identifying the type of the load (See Remark 1)
EXTID	Integer	External identification number of the aerodynamic box (See Remark 2)
INTID	Integer	Internal identification number of the aerodynamic box
AREA	Real	Area of the box
FORCE	Real	Real part of the applied normal force
PRESS	Real	Real part of the applied pressure

Created By: Many Modules

Notes:

1. The LOADTYPE is a textual key that identifies the load terms. The following values are used:
   APPLIED - User defined applied load from all disciplines except NPSAERO. For NPSAERO, the APPLIED load is equivalent to the RIGID load. The RIGID load is not stored.
   RIGID - Trimmed rigid aerodynamic load from SAERO
   FLEXIBLE - Trimmed flexible contribution to aerodynamic load from SAERO
2. The "grids" referred to by the EXTID are actually the aerodynamic box elements. Each of these elements is physically located at the centroid of a quadrilateral or triangular plate (the location of which is stored in GEOMSA or GEOMUA depending on the model).
3. The DISC flag also indicates which model is referred to by the results:
   NPSAERO refers to the nonplanar static aero model
   SAERO refers to the planar static aero model

Entity: **OCEIGS**

Entity Type: Relation

Description: Contains statistical information of complex eigenvalue analysis.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
METHOD	Text(8)	Method of complex eigenvalue extraction
BCID	Integer	Boundary condition number
NLAMA	Integer	Number of eigenvalue
NVECTOR	Integer	number of eigenvectors
NOSTRT	Integer	Number of passes through the starting points
NOMOVS	Integer	Number of starting point moves
NODCMP	Integer	Number of decomposition
ITER	Integer	Total number of iterations
ITERM	Integer	Reason for termination

Created By: Module CEIG

Entity: **OCPARM**

Entity Type: Realtion

Description: Contains the optimizer parameters and their new values for use in the ASTROS optimality criterion methods as input from the Bulk Data File.

NAME	TYPE/KEY	DESCRIPTION
PARAM	Text (8)	Name of the parameter
INTPARM	Integer	Value of integer parameters
RSPPARM	Real	Value of real parameters

Created By: Module IFP

Notes:

1. Used by the VANGO module to override default values of control parameters.

Entity: **OEIGS**

Entity Type: Relation

Description: Contains statistical information of real eigenvalue analysis.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
METHOD	Text(8)	Method of real eigenvalue extraction
NITER	Integer	Design iteration number
BCID	Integer	Boundary condition number
NLAMA	Integer	Number of eigenvalue
NVECTOR	Integer	number of eigenvectors
NEVER	Integer	Number of eigenvalue errors
NVER	integer	Number of eigenvector errors
NOSTRT	Integer	Number of passes through the starting points
NOMOVS	Integer	Number of starting point moves
NODCMP	Integer	Number of decomposition
ITER	Integer	Total number of iterations
ITERM	Integer	Reason for termination
XMAX1	Real	Maximum off diagonal mass term
ISTORE	Integer	The row number at which the maximum off diagonal mass term is located
JSTORE	Integer	The column number at which the maximum off diagonal mass term is located
IMSG	Integer	Number of off diagonal mass terms
TITLE	Text(72)	Not used
SUBTITLE	Text(72)	
LABEL	Text(72)	

Created By: Module REIG

Entity: **OEULBUCK**

Entity Type: Relation

Description: Contains Euler buckling constraint output.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
BCID	Integer	Boundary condition number
DISC	Integer	Discipline flag
ITER	Integer	Design iteration number
SID	Integer	Constraint set identification number
SUBCASE	integer	Subcase number
EID	Integer	Element identification number
ETYPE	Text(8)	Element type
BNDCON	Text(8)	Boundary condition type
PAIXAL	Real	Intermediate rsult in constraint computation
LAMBDA	Real	Extracted eigenvalue
BKMODE	Text(8)	Buckling mode
CVAL	Real	Constraint value
LENGTH	Real	Rod buckling length

Created By: Module EBKLEVAL

Entity: **OGPWG**

Entity Type: Relation

Description: Contains data from the grid point weight generation computations.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
NITER	Integer	Iteration number
BCID	Integer	Boundary condition number
GREF	Integer	Grid point identification (or zero)
XO	Real	Basic coordinates of the reference point
YO	Real	
ZO	Real	
MO	R Vector(36)	Mass matrix at the reference point
S	R Vector(9)	Principal axes relative to basic system
MX	Real	Mass in the x-axis direction
RX	R Vector(3)	x,y,z coordinates of the x-axis c.g.
MY	Real	Mass in the y-axis direction
RY	R Vector(3)	x,y,z coordinates of the y-axis c.g.
MZ	Real	Mass in the z-axis direction
RZ	R Vector(3)	x,y,z coordinates of the z-axis c.g.
INERTIA	R Vector(9)	Matrix of inertias
PINERTIA	R Vector(3)	Principal inertias about x
Q	R Vector(9)	Components of the principal axes

Created By: Module GPWG

Entity: **OGRIDDSP**

Entity Type: Relation

Description: Contains the displacements of the physical degrees of freedom that are requested for print or punch in Solution Control.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
NITER	Integer	Iteration number
BCID	Integer	The boundary condition number
DISC	Integer	Discipline flag = 1 Statics = 5 Transient = 2 Modes = 6 Frequency = 3 Steady Aero = 7 Buckling = 4 Flutter = 8 Blast
SUBCASE	Integer	Subcase identification number
DISPTYPE	Text (8)	Label identifying the type of the displacement (See Remark 1)
CMPLX	Integer	Real or complex flag = 1 if real displacement = 2 if complex displacement
GRIDID	Integer	External identification number of the physical point
SIL	Integer	Internal identification number of the physical point
FLAG	Integer	Flag indicating whether the point is a grid point or a scalar point = 0 for extra points = 1 for scalar points = 6 for structural nodes
RDISP	Real Vector (6)	Real part of the displacement
IDISP	Real Vector (6)	Imaginary part of the displacement

Created By: Many Modules

Notes:

1. The DISPTYPE is a textual key that identifies the displacement terms. The following values are used:
   DISPLACE - Displacements of the structural degrees of freedom.
   VELOCITY - Velocities of the structural degrees of freedom
   ACCEL - Accelerations of the structural degrees of freedom.

Entity: **OGRIDLOD**

Entity Type: Relation

Description: Contains the applied loads, reaction forces and other loads on the physical degrees of freedom that are requested for print or punch in Solution Control.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
NITER	Integer	Iteration number
BCID	Integer	The boundary condition number
DISC	Integer	Discipline flag = 1 Statics = 5 Transient = 2 Modes = 7 Buckling = 3 Steady Aero = 8 Blast
SUBCASE	Integer	Subcase identification number
LOADTYPE	Text (8)	Label identifying the type of the load (see remark 1)
CMPLX	Integer	Real or complex flag = 1 if real load = 2 if complex load
GRIDID	Integer	External identification number of the physical point
SIL	Integer	Internal identification number of the physical point
FLAG	Integer	Flag indicating whether the point is a grid point or a scalar point = 0 for extra point = 1 for scalar point = 6 for structural nodes
RFORCE	Real	Real part of the applied load
IFORCE	Real	Imaginary part of the applied load

Created By: Many Modules

Notes:

1. The LOADTYPE is a textual key that identifies the load terms. The following values are used:
   - APPLIED - User defined applied load from all disciplines. For SAERO, the APPLIED load is computed and stored as the sum of RIGID, FLEXIBLE and INERTIA loads.
   - RIGID - Trimmed rigid aerodynamic load from SAERO
   - FLEXIBLE - Trimmed flexible contribution to aerodynamic load from SAERO
   - INERTIA - Inertia load contribution from SAERO and STATICS with inertia relief
   - SPC - SPC reaction forces for STATICS, SAERO, MODES, TRANSIENT and FREQUENCY.

Entity: **OLOCALDV**

Entity Type: Relation

Description: Contains the local design variable values that are requested for print or punch in Solution Control.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
NITER	Integer	Iteration number
ETYPE	Text (8)	Element type, one of the following: BAR CONM2 ELAS MASS QDMEM1 QUAD4 ROD SHEAR TRIA3 TRMEM
EID	Integer > 0	Element identification number
LAYRNUM	Integer	Layer number (=0 if noncomposite)
DVSYMBL	Text(8)	Designed dimension symbol (D1, D2, D3, D4, D5, D6, D7, D8, D9, D10 for BAR element cross section dimension; A for element area, T for element thickness; M for element mass; K for element stiffness.)
T	Real	Local design variable value (See Remark 2)
I1	Real	1st plane moment of inertia for BAR elements
I2	Real	2nd plane moment of inertia for BAR elements

Created By: Module ACTCON

Notes:

1. Any local design variable that are requested for print or punch in Solution Control at any iteration will be stored in this relation.
2. For each element type, T, I1 and I2 have different meanings
   - BAR - T is element cross-sectional area
     I1 and I2 are related moments of inertia
   - CONM2 - T is concentrated mass value
     I1 and I2 are not used
   - ELAS - T is spring stiffness
     I1 and I2 are not used
   - MASS - T is mass value
     I1 and I2 are not used
   - QDMEM1 - T is element or layer thickness
     I1 and I2 are not used
   - QUAD4 - T is element or layer thickness
     I1 and I2 are not used

ROD	-	T is element cross sectional area I1 and I2 are not used
SHEAR	-	T is element thickness I1 and I2 are not used
TRIA3	-	T is element or layer thickness I1 and I2 are not used
TRMEM	-	T is element or layer thickness I1 and I2 are not used

Entity: **OMIT**

Entity Type: Relation

Description: Contains the definition of the degrees of freedom that the user wishes to omit from the analysis through matrix reduction.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
GRID1	Integer > 0	Grid or scalar point id
COMPNTS1	Integer ≥ 0	Component of GRID1 to be omitted

Created By: Module IFP

Notes:

1. Used by the MKUSET module to build the USET relation.

Entity: **OMIT1**

Entity Type: Relation

Description: Contains the definition of the degrees of freedom that the user wishes to omit from the analysis through matrix reduction.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
COMPNTS1	Integer ≥ 0	Component of GRID1 to be omitted
GRID1	Integer > 0	Grid or scalar point id

Created By: Module IFP

Notes:

1. Used by the MKUSET module to build the USET relation.

Entity: **OPNLBUCK**

Entity Type: Relation

Description: Contains panel buckling constraint output.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
BCID	Integer	Boundary condition number
DISC	Integer	Discipline flag
ITER	Integer	Design iteration number
SID	Integer	Constraint set identification number
SUBCASE	integer	Subcase number
EID	Integer	Element identification number
ETYPE	Text(8)	Element type
NX	Real	Element force
NY		
NXY		
SHIFT	Real	Shifting points adopted by power method
LAMBDA	Real	Extracted eigenvalue
BKMODE	Text(8)	Buckling mode
CVAL	Real	Constraint value
LENGTH	Real	Panel buckling length
WIDTH	Real	Panel buckling width

Created By: Module PBKLEVAL

Entity: **OTL**

Entity Type: Unstructured

Description: Contains a list of output times for each time step set.

Record:

1. Contains a list of the LIDs of the time step sets in the Bulk Data file.
i. Contains the output time list for the $(i-1)^{th}$ set ID.

Created By: Module PFBULK

Notes:

1. This entity is used in the OFPxxx modules.

Entity: **OPTIMIZE**

Entity Type: Relation

Description: Contains the optimization-dependent solution control requests as input in the solution control packet.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
CGRAPRNT	Integer vector (20)	Constraint gradient print selection WORD 1 Print set identification number > 0, or = 0 NONE = –1 ALL = –2 LAST = –3 ACTIVE WORD 2 Punch set identification number WORD 3 Print form = 0 Rectangular = 1 Polar WORD 4 Punch form WORD 5 Print frequency set identification number WORD 6 Punch frequency set identification number WORD 7 Print iteration set identification number WORD 8 Punch iteration set identification number WORD 9 Print mode set identification number WORD 10 Punch mode set identification number WORD 11 Print time set identification number WORD 12 Punch time set identification number WORD 13-20 unused
OBJID	Integer	Objective function identification number
DCONFID	Integer	Subcase-independent user function constraint identification number
OBJDIR	Text (4)	MIN for MINIMIZE MAX for MAXIMIZE
DCONPRNT	Integer vector (20)	Design constraint print selection
GDESPRNT	Integer vector (20)	Global design variable print selection
KSNSPRNT	Integer vector (20)	Element stiffness senstivity print selection

NAME	TYPE/KEY	DESCRIPTION
LDESPRNT	Integer vector (20)	Local design variable print selection
MSNSPRNT	Integer vector (20)	Element mass senstivity print selection
OGRAPRNT	Integer vector (20)	Objective function gradient print selection
BULKPRNT	Integer vector (20)	Design model Bulk Data punch selection
HISTPRNT	Integer	Design iteration history print toggle
TITLE	Text (72)	User label TITLE
SUBTITLE	Text (72)	User label SUBTITLE
LABEL	Text (72)	User label LABEL

Created By: Module Solution

Notes:

1. The format of the CGRAPRNT vector is typical of the format of all the print selection vectors. Additionally, the format for the print set Identification number in the CGRAPRNT vector is typical of that of the other set Identification numbers in the vector.
2. The CASE, JOB and OPTIMIZE relation entities together contain the solution control requests as input in the solution control packet. CASE contains the case-dependent parameters, JOB contains the case-independent requests and OPTIMIZE contains the optimization-dependent requests.

Entity: **PA**

Entity Type: Matrix

Description: External loads applied in the a-set derived from partitions of PF (see PG).

Entity: **PAA**

Entity Type: Matrix

Description: Rigid body aerodynamic load vectors derived from partitions of PAF (see PAF).

Entity: **PAERO1**

Entity Type: Relation

Description: Contains a list of associated bodies for panels used in Doublet-Lattice aerodynamics.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
PID	Integer > 0	Property identification number
BODIES	Integer Array (6)	Array attribute containing the identifications of associated bodies

Created By: Module IFP

Notes:

1. The BODIES identification numbers refer to CAERO2 relation tuples.

Entity: **PAERO2**

Entity Type: Relation

Description: Contains the definition of the cross-sectional properties of Doublet-Lattice aerodynamic bodies as input from the bulk data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
PID	Integer > 0	Property identification
ORIENT	Text (8)	Type of motion allowed for the body
WIDTH	Real > 0.0	Reference half width for the body
AR	Real ≥ 0.0	Aspect ratios for the body
LRSB	Integer > 0	AEFACT identification number containing the half widths of slender bodies
LRIB	Integer > 0	AEFACT identification number containing the half widths of interference bodies
LTH1	Integer ≥ 0	AEFACT identification number that has the first array of theta values
LTH2	Integer ≥ 0	AEFACT identification number that has the second array of theta values
THI1	Integer ≥ 0	First interference element using the LTH1 theta distribution
THN1	Integer ≥ 0	Last interference element using the LTH1 theta distribution
THI2	Integer ≥ 0	First interference element using the LTH2 theta distribution
THN2	Integer ≥ 0	Last interference element using the LTH2 theta distribution
THI3	Integer ≥ 0	First interference element after THN2 that uses the LTH1 theta distribution
THN3	Integer ≥ 0	Last interference element after THN2 that uses the LTH1 theta distribution

Created By: Module IFP

Notes:

Entity: **PAERO6**

Entity Type: Relation

Description: Contains the definition of analysis parameters for bodies in the aerodynamic model as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
BCID	Integer > 0	Body component identification number
ACMPNT	Text (8)	Component type (i.e. FUSEL)
CP	Integer ≥ 0	Coordinate system in which geometry inputs are given
GROUP	Integer ≥ 0	Group identification number
NRAD	Integer ≥ 0	Number of equal radial cuts used to define body panels
LRAD	Integer ≥ 0	AEFACT set identification number for the angular locations of body panels
AXIAL	Integer ≥ 0	AEFACT set identification number for the axial locations of body panels

Created By: Module IFP

Entity: **PAF**

Entity Type: Matrix

Description: Rigid body load vectors multiplied by dynamic pressure.

Matrix Form: See AIRFRC for the dimensions.

Created By: MAPOL

Notes:

1. This matrix is the dynamic pressure times AIRFRC.
2. The MAPOL sequence supports the following partitions of the PAF matrix (see the Theoretical Manual for the explicit formation of these submatrices):

$$\mathbf{PAF} \rightarrow \begin{bmatrix} \mathbf{POARO} \\ \mathbf{PAA} \end{bmatrix}$$

$$\mathbf{PAA} \rightarrow \begin{bmatrix} \mathbf{PAL} \\ \mathbf{PARBAR} \end{bmatrix}$$

Entity: **PAL**

Entity Type: Matrix

Description: A partition of PAA (see PAF).

Entity: **PAR**

Entity Type: Subscripted Matrix

Description: An intermediate matrix formed during the performance of an aeroelastic trim analysis.

Matrix Form: The number of rows is equal to the number of degrees of freedom in the l-set while the number of columns is equal to the number of rigid body load vectors from AIRFRC.

Created By: MAPOL using GFBS

Notes:

1. PAR is the solution of:

   [KA11] [PAR] = [P1]

2. Since PAR is needed in the sensitivity analysis, it is subscripted by boundary condition.

Entity: **PARBAR**

Entity Type: Matrix

Description: A partition of the PAA (see PAF).

Entity: **PARL**

Entity Type: Subscripted Matrix

Description: Contains the partitioning vector to partition those degrees of retained for analysis (a-set) into those reduced out (r-set) and those left over (l-set).

Matrix Form: A variable-sized single precision column vector having one row for each degree of freedom retained for analysis. Degrees of freedom in the reduce set are denoted by a real 0.0 and those left over by a real 1.0.

Created By: Module MKUSET

Notes:

1. The dimension of this subscripted matrix must be large enough for all optimization and analysis boundary conditions.
2. This vector is modified by the GDR modules if Generalized Dynamic Reduction is used.

Entity: **PBAR**

Entity Type: Relation

Description: Contains the property definition for the BAR element as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
PID	Integer > 0, key	Property identification number
MID1	Integer > 0	Material property identification number of the MAT1 tuple
AREA	Real > 0.0	Element cross-sectional area
I1	Real	Area moment of inertia in plane 1
I2	Real	Area moment of inertia in plane 2
TORSION	Real	Torsional constant
NSM	Real ≥ 0.0	Element nonstructural mass
TMIN	Real	Minimum cross-sectional area in design
C1, C2, D1, D2	Real	Element stress recovery coefficients
E1, E2, F1, F2	Real	
KFACT1	Real	Area factor for shear (plane 1)
KFACT2	Real	Area factor for shear (plane 2)
I12	Real	Area product of inertia
R1SQR	Real	Multiplicative factor to determine I1 in design
R2SQR	Real	Multiplicative factor to determine I2 in design
ALPHA	Real	Exponential power associated with the design variable.

Created By: Module IFP

Entity: **PBAR1**

Entity Type: Relation

Description: Contains the property definition for the BAR element as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
PID	Integer > 0, key	Property identification number
MID	Integer > 0	Material property identification number of the MAT1 tuple
SHAPE	Text(8)	Element cross-section shape
D1	Real ≥ 0.0	Element cross-section dimension 1
D2	Real ≥ 0.0	Element cross-section dimension 2
D3	Real ≥ 0.0	Element cross-section dimension 3
D4	Real ≥ 0.0	Element cross-section dimension 4
D5	Real ≥ 0.0	Element cross-section dimension 5
D6	Real ≥ 0.0	Element cross-section dimension 6
D7	Real ≥ 0.0	Element cross-section dimension 7
D8	Real ≥ 0.0	Element cross-section dimension 8
D9	Real ≥ 0.0	Element cross-section dimension 9
D10	Real ≥ 0.0	Element cross-section dimension 10
NSM	Real ≥ 0.0	Non structural mass

Created By: Module IFP

Entity: **PCAS**

Entity Type: Unstructured

Description: Identifies active constraints for the current boundary condition.

Entity Structure: A single record of integers whose length is equal to the number of constraints active in the current boundary condition.

Created By: Module ABOUND

Notes:

1. There is one integer for each active constraint. The integer is set to the subcase number of the constraint (see the CONST relation).

Entity: **PCOMP**

Entity Type: Relation

Description: Contains the property definitions for a multiple ply composite material laminate as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
PID	Integer > 0	Property identification number
ZO	Real	The distance from the plane of the grid points to the bottom surface
NSM	Real ≥ 0.0	nonstructural mass per unit area
SBOND	Real > 0.0	Allowable shear stress of bonding material
FAILCRIT	Text (8)	Theory used to predict failure
TMIN	Real	Minimum layer thicknesses for design
LOPT	Text (8)	Laminate generation option
MIDI	Integer ≥ 0	Ply material identification
THICKI	Integer ≥ 0	Ply thickness
THETAI	Real	Ply material orientation angle
SOUTI	Text (8)	Flag for stress output

Created By: Module IFP

Notes:

1. This relation will contain one tuple for each ply in each unique PID.

Entity: **PCOMPS**

Entity Type: Unstructured

Description: Contains one record for each PCOMPi Bulk Data type entry. Data includes the PCOMPi entry and its intrinsic laminate property data.

Entity Structure:

Record	Word	Type	Description
1	1-2	Text	PCOMP
	3	Integer > 0	PID-Property identification number
	4	Integer > 0	N-Number of layers
	5(11+4*N)	RSP	Remainder of PCOMP data
	(12+4*N)-(31*N+11)	RSP	Layer Property data
	(31*N+12)-(31*N+13)	RSP	Laminate Bending Inertia
	(31*N+14)-(31*N+15)	RSP	Laminate Neutral Surface Location
Words 3 through 31*N+15 are repeated for each PCOMP Bulk Data entry.			
2	1-2	Text	PCOMP1
	3	Integer > 0	PID-Property identification number
	4	Integer > 0	N-Numbers of layers
	5-(12+N)	RSP	Remainder of PCOMP1 data
	(13+N)-(37+N)	RSP	Layer property data
	(38+N)-(39+N)	RSP	Laminate Bending Inertia
	(40+N)-(41+N)	RSP	Laminate Neutral Surface Location
Words 3-41+N are repeated for each PCOMP1 Bulk Data entry.			
3	1-2	Text	PCOMP2
	3	Integer > 0	PID-Property identification number
	4	Integer > 0	N-Number of layers
	5-(11+2*N)	RSP	Remainder of PCOMP2 data
	(12+2*N)-(36+2*N)	RSP	Layer property data
	(37+2*N)-(38+2*N)	RSP	Laminate Bending Inertia
	(39+2*N)-(40+2*N)	RSP	Laminate Neutral Surface Location
Words 3-40+2*N are repeated for each PCOMP2 Bulk Data entry.			

Created By: Module EMG

Entity: **PCOMP1**

Entity Type: Relation

Description: Defines the property of a n-ply laminated composite material where all plies are composed of the same material and are of equal thickness.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
PID	Integer > 0	Property identification number
Z0	Real	Offset of the element reference plane from the plane of grid points
NSM	Real > 0.0	nonstructural mass per unit area
SBOND	Real > 0.0	Allowable shear stress of the bending material.
FAILCRIT	Text (8)	Failure theory to predict ply failure
TMIN	Real > 0.0	Minimum layer thickness for design
MID	Integer > 0	Ply material identification
LOPT	Text (8)	Lamination generation option
THICK	Real > 0.0	Ply thickness
THETAI	Real	Ply material orientation angle

Created By: Module IFP

Notes:

1. This relation will contain one tuple for each ply for each unique PID.

Entity: **PCOMP2**

EntityType: Relation

Description: Defines the properties of a n-ply laminated composite material where all plies are of the same material.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
PID	Integer > 0	Property identification number
Z0	Real	Offset of the element reference plane from the plane of grid points.
NSM	Real > 0.0	Nonstructural mass per unit area
SBOND	Real > 0.0	Allowable shear stress of the bonding material.
FAILCRIT	Text (8)	Failure theory to predict ply failure
TMIN	Real > 0.0	Minimum layer thickness for design
MID	Integer> 0.0	Ply material identification
LOPT	Text (8)	Lamination generation option
THICKI	Real > 0.0	Ply thickness
THETAI	Real	Ply material orientation angle

Created By: Module IFP

Notes:

1. The relation will contain one tuple for each ply for each unique PID.

Entity: **PDF**

Entity Type: Matrix

Description: Applied loads matrix for frequency response analysis.

Matrix Form: Complex matrix with one column for each frequency at which frequency response results are to be computed. This matrix is applicable for both the direct and modal methods of solution so that the number of rows equal to the number of degrees of freedom in the d- or h-sets, depending on the method of solution.

Created By: Module DYNLOAD

Notes:

1. This matrix is also for applied gust loads if the gust discipline option of frequency response is selected.

Entity: **PDLIST**

EntityType: Relation

Description: Contains the panel buckling constraint sensitivity parameters.

NAME	TYPE/KEY	DESCRIPTION
NITER	Integer > 0	Iteration number
BCID	Integer > 0	Boundary condition identification number
DISFLAG	Integer > 0	Discipline type flag from CASE relation (where appropriate)
PNUM	Integer > 0	Pointer number from CONST for sensitivity computation
SID	Integer > 0	Set identification number of panel buckling constraints
UPLOW	Text (8)	Constraint type (Upper/Lower)
POWER	Real	Internal constraint formulation factor
PID	Integer	Element property identification number
PTYPE	Text(8)	Element property type
MID	Integer	Element material identification number
MTYPE	Text(8)	Element property type (PCOMPS/PSHELL)
THICK	Real	Element thickness
DCVAL	Real	Intermediate result in constraint computation
LAMBDA	Real	Extracted eigenvalue
PD11	Real	Intermediate results for sensitvity computation
PD12		
PD13		
PD22		
PD23		
PD33		

Created By: Module PBKLEVAL

Entity: **PDT**

Entity Type: Matrix

Description: Applied loads matrix for transient response analysis.

Matrix Form: Complex matrix with one column for each frequency at which transient response results are to be computed. This matrix is applicable for both the direct and modal methods of solution so that the number of rows equal to the number of degrees of freedom in the d- or h-sets, depending on the method of solution.

Created By: Module DYNLOAD

Entity: **PELAS**

Entity Type: Relation

Description: Contains the property data for scalar spring elements as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
PID	Integer, key	Property identification number
K	Real	Spring constant
DAMPCOEF	Real	Damping coefficient
STRSCOEF	Real	Stress coefficient
TMIN	Real	Minimum spring constant value for design

Created By: Module IPF

Entity: **PF**

Entity Type: Matrix

Description: External loads in the f-set derived from partitions of PN (see PG).

Entity: **PFGLOAD**

Entity Type: Matrix

Description: Applied loads matrix on the physical degrees of freedom for the frequency dependent loads in the current boundary condition.

Matrix Form: Complex rectangular matrix with one row for each physical degree of freedom and one column for each frequency step in each frequency analysis in the current boundary condition.

Created By: Module DYNLOAD

Notes:

1. This matrix is formed only if the LOAD print request for the FREQUENCY discipline is set for the current boundary condition.

Entity: **PFOA**

Entity Type: Subscripted Matrix

Description: Contains the partitioning vector to partition the free degrees of freedom (f-set) into the omitted degrees of freedom (o-set) and those retained for analysis (a-set).

Matrix Form: A variable-sized single-precision column vector containing one row for every free degree of freedom. Degrees of freedom in the o-set are denoted by real 0.0 and those in the a-set by real 1.0.

Created By: Module MKUSET

Notes:

1. The dimension of this subscripted matrix must be large enough for all optimization and analysis boundary conditions.
2. This matrix is modified by the GDR modules if Generalized Dynamic Reduction is used.

Entity: **PG**

Entity Type: Matrix

Description: Contains the global loads matrix for the current boundary condition.

Matrix Form: A variable-size matrix having one row for each structural degree of freedom in the model and one column for each load condition in the current boundary condition.

Created By: Module GTLOAD

Notes:

1. This matrix is flushed and re-formed for each boundary condition in the problem.
2. The MAPOL sequence supports the following partitions of the PG matrix (see Theoretical Manual for the explicit formation of these submatrices).

$$\mathbf{PG} \rightarrow \begin{bmatrix} \varphi \\ \mathbf{PN} \end{bmatrix} \qquad \mathbf{PN} \rightarrow \begin{bmatrix} \mathbf{PS} \\ \mathbf{PF} \end{bmatrix}$$

$$\mathbf{PF} \rightarrow \begin{bmatrix} \mathbf{PO} \\ \mathbf{PA} \end{bmatrix} \qquad \mathbf{PA} \rightarrow \begin{bmatrix} \mathbf{PR} \\ \mathbf{PLBAR} \end{bmatrix}$$

Entity: **PGA**

Entity Type: Matrix

Description: Partitioning vector for active load cases.

Matrix Form: One column with the numbers of rows equal to the number of subcases for the current boundary condition.

Created By: Module ABOUND

Notes:

1. Active subcases are designated by a value of 1.0, inactive subcases by 0.0.

Entity:	**PGMN**
Entity Type:	Subscripted Matrix
Description:	Contains the partitioning vector to partition the structural degrees of freedom (g-set) into the dependent multi-point constraint set (m-set) and the independent set (n- set).
Matrix Form:	A variable-sized single precision column vector containing one row for each structural degree of freedom in the model. Degrees of freedom in the m-set are denoted by real 0.0 and those in the n-set by real 1.0.
Created By:	Module MKUSET

Entity:	**PHIA**
Entity Type:	Matrix
Description:	Contains the eigenvectors in the analysis degrees of freedom for each vector computed.
Matrix Form:	A variable-sized vector having one column for each computed eigenvector and one row for each degree of freedom in the analysis set for the current boundary condition.
Created By:	Module REIG

Notes:

1. See PHIG for data recovery.

Entity:	**PHIB**
Entity Type:	Matrix
Description:	Matrix of mode shapes used in nuclear blast response.
Matrix Form:	Real rectangular matrix with one row for each a-set degree of freedom and one column for each retained mode.
Created By:	MAPOL

Notes:

1. PHIB is obtained by merging PHIR and PHIE.

Entity:	**PHIE**
Entity Type:	Matrix
Description:	Matrix of elastic mode shapes used in nuclear blast response.
Matrix Form:	Real rectangular matrix with one row for each a-set degree of freedom and one column for each retained elastic mode.
Created By:	MAPOL

Entity: **PHIF**

Entity Type: Matrix

Description: Normal modes in the f-set recovered from PHIA and PHIO (see PHIG).

Entity: **PHIG**

Entity Type: Subscripted Matrix

Description: Contains the eigenvectors in the global set computed in the REIG module.

Matrix Form: A variable-sized matrix having one column for each eigenvector computed and one row for each structural degree of freedom.

Created By: MAPOL

Notes:

1. The MAPOL sequence recovers this matrix in the following order:

```
[PHIO] = [GSUBO] [PHIA]

[ PHIA ]
[ PHIO ]  →  PHIF

[  YS  ]
[ PHIF ]  →  PHIN

[UM*] = [TMN] [PHIN]

[  UM  ]
[ PHIN ]  →  PHIG
```

*UM contains modes in the m-set. The entity is reused in the MAPOL sequence.

Entity: **PHIKH**

Entity Type: Matrix

Description: Normal mode shapes splined to the aerodynamic panels.

Matrix Form: Real rectangular matrix with one row for each aerodynamic degree of freedom and one column for each normal mode.

Created By: Module QHHLGEN

Entity: **PHIN**

Entity Type: Matrix

Description: Modes in the n-set, recovered from PHIF (see PHIG).

Entity: **PHIO**

Entity Type: Matrix

Description: Mode shapes for omitted degrees of freedom (see PHIG).

Entity:	**PHIOK**
Entity Type:	Matrix
Description:	Approximate mode shapes produced by generalized dynamic reduction.
Matrix Form:	Real rectangular matrix with one row for each o-set degree of freedom and one column for each approximate mode shape.
Created By:	Module GDR2
Notes:	1. This matrix is computed for generalized dynamic reduction and only if there are k-set degrees of freedom.

Entity:	**PHIR**
Entity Type:	Matrix
Description:	Matrix of rigid body shapes used in nuclear blast response.
Matrix Form:	Real rectangular matrix with one row for each a-set degree of freedom and one column for each retained rigid body mode.
Created By:	MAPOL
Notes:	1. This matrix is created by performing a ROWMERGE using matrices D and ID2 and partiton vector MPART.

Entity: **PIHEX**

Entity Type: Relation

Description: Contains the property data for an isoparametric hexahedron element as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
PID	Integer > 0, key	Property identification number
MID	Integer > 0	Material identification number
CID	Integer ≥ 0	Identification number of the coordinate system in which the material referenced by MID is defined
NIP	Integer = 2, 3, 4	Number of integration points along each edge of the element
AR	Real > 1.0	Maximum aspect ratio (ratio of longest to shortest edge) of the element
ALFA	Real, 0.0≤ALFA≤180.0	Maximum angle in degrees between the normals of two subtriangles comprising a quadrilateral face
BETA	Real, 0.0≤BETA≤180.0	Maximum angle in degrees between the vector connecting a corner point to an adjacent midside point and the vector connecting that midside point and the other midside or corner point

Created By: Module IFP

Notes:

Entity: **PLBAR**

Entity Type: Matrix

Description: A partition of matrix PA (see PG).

Entity: **PLIST**

Entity Type: Relation

Description: Contains the property types and identification numbers associated with a design variable.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
LINKID	Integer > 0, key	Design variable identification
PTYPE	Text (8)	Property relation identifier
PID1	Integer > 0	Property identification

Created By: Module IFP

Notes:

1. The PTYPE is the name of the relation in which the PID associated with the design variable is found.
2. This relation contains one tuple for each PID associated with each PTYPE listed in each unique LINKID.
3. Allowable PTYPE entries are:

PROD	PCOMP, PCOMP1, PCOMP2
PSHEAR	PMASS
PQDMEM	PSHELL
PTRMEM	PELAS
PBAR	

Entity: **PLISTM**

Entity Type: Relation

Description: Contains the property types and identification numbers associated with a design variable.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
LINKID	Integer > 0, key	Design variable identification
PTYPE	Text (8)	Property relation identifier
PID1	Integer > 0	Property identification
DVSYMBL	Text(8)	Designed dimension symbol

Created By: Module IFP

Notes:

1. The PTYPE is the name of the relation in which the PID associated with the design variable is found.
2. This relation contains one tuple for each PID associated with each PTYPE listed in each unique LINKID.
3. Allowable PTYPE entries are:

PROD	PCOMP, PCOMP1, PCOMP2
PSHEAR	PMASS
PQDMEM	PSHELL
PTRMEM	PELAS
PBAR	

4. Allowable DVSYMBL are:
   - D1, D2, D3, D4, D5, D6, D7, D8, D9, D10 for BAR element cross section dimensions;
   - A for element cross section area;
   - T for element thickness;
   - M for element mass;
   - K for element thickness.

Entity: **PLOAD**

Entity Type: Relation

Description: Contains the load information defined over a triangular or quadrilateral region as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
SCALE	Real	Scale factor
GRID1, GRID2	Integer > 0	Grid points defining region of load application
GRID3, GRID4	Integer > 0	

Created By: Module IFP

Notes:

1. The GRID4 entry is zero if a triangular region is defined.

Entity: **PLOAD2**

Entity Type: Relation

Description: Contains the uniform static pressure load information defined over a triangular or quadrilateral region as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
SCALE	Real	Scale factor
EID	Integer > 0	Element identification number

Created By: Module IFP

Entity: **PLOAD4**

Entity Type: Relation

Description: Contains the load information defined over a triangular or quadrilateral region as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
EID1	Integer > 0	Element identification number
P1	Real	Pressure at Grid point 1
P2	Real	Pressure at Grid point 2
P3	Real	Pressure at Grid point 3
P4	Real	Pressure at Grid point 4
THRU	Text(4)	"THRU" string for range definition of element IDs
EID2	Integer ≥ 0	The last element identification number in a range definition
CID	Integer ≥ 0	Coordinate system identification number
V1	Real	Vector components in system CID that defines the direction of the GRID point loads generated by the pressure
V2		
V3		

Created By: Module IFP

Entity: **PLYLIST**

Entity Type: Relation

Description: Contains a list of composite layers as input in the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer > 0	Set identification number
PLY	Integer > 0	Ply number

Created By: Module IFP

Notes:

1. This relation contains one tuple for each ply in each set.

Entity: **PMASS**

Entity Type: Relation

Description: Contains the mass value of a scalar mass element as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
PID	Integer > 0, key	Property identification number
MASS	Real	Mass value
TMIN	Real > 0.0	Minimum mass value for design

Created By: Module IFP

Entity: **PMAXT**

Entity Type: Matrix

Description: Contains the maximum thickness design variable based on the user's defined maximum (laminate) thickness.

$$[t] = [PMAXT]^{t}[v] + [VFIXD]$$

Matrix Form: A variable-sized single precision matrix having one column for each shape function designed laminate or element and one row for each global design variable. The terms in PMAXT are the sum of the PTRANS columns associated with one laminate (if composite).

Created By: Module MAKEST

Notes:

1. If a layered composite has some undesigned laminae, the VFIXD entity contains the terms needed to calculate the fixed contribution.
2. If no shape function linking is used, this matrix will have no columns.

Entity: **PMINT**

Entity Type: Matrix

Description: Contains the minimum thickness variable linking terms based on the user's defined minimum (laminate) thickness.

$$[t] = [PMINT]^{t}[v]$$

Matrix Form: A variable-size single precision matrix that has one column for each element designed by shape function linking and one row for each global design variable. The terms in PMINT are the PTRANS column for the shape function designed element divided by the user input minimum (laminate) thickness.

Created By: Module MAKEST

Notes:

1. If no shape function linking is used, this matrix will have no columns.

Entity: **PN**

Entity Type: Matrix

Description: External loads applied in the n-set derived from PG (see PG).

Entity: **PNSF**

Entity Type: Subscripted Matrix

Description: Contains the partitioning vector to partition the independent degrees of freedom (n-set) into the dependent single point constraint set (s-set) and the free degrees of freedom (f-set).

Matrix Form: A variable-sized single precision column vector containing one row for each independent degree of freedom. Degrees of freedom in the s-set are denoted by real 0.0 and those in the f-set by real 1.0.

Created By: Module MKUSET

Entity: **PO**

Entity Type: Matrix

Description: A partition of the PF matrix (see PG).

Entity: **POARO**

Entity Type: Subscripted Matrix

Description: Matrix of aerodynamic "unit" loads applied to omitted degrees of freedom.

Matrix Form: Real rectangular matrix with one row for each o-set degree of freedom and the same number of columns as the AIRFRC matrix.

Created By: MAPOL

Notes:

1. The Matrix may be required in the static aeroelastic sensitivity analysis and is therefore subscripted by the boundary condition.

Entity: **PQDMEM1**

Entity Type: Relation

Description: Contains the properties of the isoparametric quadrilateral membrane element as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
PID	Integer > 0, key	Property identification number
MID1	Integer > 0	Material identification number
THICK	Real> 0.0	Element thickness
NSM	Real ≥ 0	Element nonstructural mass
TMIN	Real ≥ 0	Minimum thickness for design

Created By: Module IFP

Entity: **PR**

Entity Type: Matrix

Description: A partition of the PA matrix (see PG).

Entity: **PRAS**

Entity Type: Unstructured

Description: Identifies active user function constraints required responses for the current boundary condition.

Entity Structure: A single record of integers whose length is equal to the number of responses required by active user function constraints in the current boundary condition.

Created By: Module ABOUND

Notes:

There is one integer for each active constraint. The integer is set to the subcase number of the constraint.

Entity: **PROD**

Entity Type: Relation

Description: Contains the property data for ROD elements as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
PID	Integer > 0, key	Property identification number
MID1	Integer > 0	Material identification number of a MAT1 tuple
AREA	Real ≥ 0.0	Element cross sectional area
TORSION	Real ≥ 0.0	Torsional constant
STRSCOEF	Real	Stress recovery coefficient
NSM	Real ≥ 0.0	Element nonstructural mass
TMIN	Real ≥ 0.0	Minimum cross-sectional area for design

Created By: Module IFP

Entity: **PS**

Entity Type: Matrix

Description: A partition of the PN matrix (see PG).

Entity: **PSHEAR**

Entity Type: Relation

Description: Contains the property data for the shear panel as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
PID	Integer > 0, key	Property identification number
MID1	Integer > 0	Material identification number
THICK	Real > 0	Element thickness
NSM	Real ≥ 0.0	Element nonstructural mass
TMIN	Real ≥ 0.0	Minimum thickness for design

Created By: Module IFP

Entity: **PSHELL**

Entity Type: Relation

Description: Contains the membrane, bending, shear and coupling properties of thin two-dimensional elements as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
PID	Integer > 0, key	Property identification number
MID1	Integer > 0	Membrane material id
THICK	Real > 0	Element default thickness
MID2	Integer ≥ 0	Bending material id
BENDSTIF	Real	Bending stiffness parameter
MID3	Integer ≥ 0	Transverse shear material id
TRNSVRS	Real	Transverse shear thickness divided by the membrane thickness
NSM	Real ≥0.0	Element nonstructural mass
FZ1, FZ2	Real	Fiber distances for stress computation
MID4	Real ≥ 0	Membrane-bending coupling material identification
CID2	Integer	Material coordinate system identification number
THETAM	Real	Material orientation angle
CIDS	Integer	Stress recovery coordinate system
THETAS	Real	Stress recovery orientation angle
OFFST1	Integer	Offset of the mid plane from the plane of the grid points
TMIN	Real ≥ 0.0	Minimum thickness for design

Created By: Module IFP

Entity: **PTGLOAD**

Entity Type: Matrix

Description: Applied loads matrix on the physical degrees of freedom for the time dependent loads in the current boundary condition.

Matrix Form: Real rectangular matrix with one row for each physical degree of freedom and one column for each time step in each transient analysis in the current coundary condition.

Created By: Module DYNLOAD

Notes:

1. This matrix is formed only if the LOAD print request for the transient discipline is set for the current boundary condition.

Entity: **PTRANS**

Entity Type: Matrix

Description: Contains the linking information for design variables if the model has design variables defined.

$$[t] = [PTRANS]^{t}[v]$$

Matrix Form: A variable-sized single precision matrix having one column for each local design variable and one row for each global design variable.

Created By: Module MAKEST

Notes:

1. This matrix is empty if the model contains no design variables.
2. A column of PTRANS is the sensitivity of the local variable to the global variable.

Entity: **PTRMEM**

Entity Type: Relation

Description: Contains the property data for the constant strain triangle as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
PID	Integer > 0, key	Property identification number
MID1	Integer > 0	Material identification number of MAT1 or MAT2 tuple
THICK	Real > 0.0	Element thickness
NSM	Real ≥ 0.0	Element nonstructural mass
TMIN	Real ≥ 0.0	Minimum thickness for design

Created By: Module IFP

Entity:	**P1**
Entity Type:	Matrix
Description:	Applied loads matrix created when there are unrestrained structural degrees of freedom.
Matrix Form:	Real, rectangular matrix with one row for each a-set degree of freedom and one column for each subcase or column in the PAF matrix, depending on whether a static analysis or a static aeroelastic analysis is being performed.
Created By:	MAPOL

Entity:	**P2**
Entity Type:	Matrix
Description:	Applied loads matrix created when there are unrestrained structural degrees of freedom.
Matrix Form:	Real, rectangular matrix with one row for each r-set degree of freedom and one column for each subcase or column in the PAF matrix, depending on whether a static analysis or a static aeroelastic analysis is being performed.
Created By:	MAPOL

Entity: **QDMM1EST**

Entity Type: Relation

Description: Contains the element summary data for the isoparametric quadrilateral membrane element.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0	Element identification number
PID	Integer > 0	Element property identification number
PTYPE	Text (8)	Element property type
LAYRNUM	Integer ≥ 0	Composite layer number
SIL1	Integer > 0	Internal grid point id
SIL2	Integer > 0	
SIL3	Integer > 0	
SIL4	Integer > 0	
CID	Integer ≥ 0	Coordinate system defining material axis
THETA	Real	Material orientation angle for anisotropic material behavior
MID1	Integer ≥ 0	Material id of MAT1 or MAT2 tuple
THICK	Real ≥ 0.0	Element thickness
NSM	Real ≥ 0.0	Element nonstructural mass
COORD1	Integer ≥ 0	External coordinate system id for displacements at SIL1
X1, Y1, Z1	Real	Basic coordinates of SIL1
COORD2	Integer ≥ 0	External coordinate system id for displacements at SIL2
X2, Y2, Z2	Real	Basic coordinates of SIL2
COORD3	Integer ≥ 0	External coordinate system id for displacements at SIL3
X3, Y3, Z3	Real	Basic coordinates of SIL3
COORD4	Integer ≥ 0	External coordinate system id for displacements at SIL4
X4, Y4, Z4	Real	Basic coordinates of SIL4
SCON	Integer	Stress constraint flag
DESIGN	Integer	Design flag

NAME	TYPE/KEY	DESCRIPTION
STHRM	Real Array (3)	Thermal stress terms for the constrained element
STHRMA	Real Array (3)	Thermal strain terms for the constrained element
TREFTP	Integer ≥ 0	Pointer to the TREF entity for thermal loads/stress evaluation of the designed element
NLFLAG	Integer ≥ 0	Nonlinear design variable flag = 0 linear = 1 NSM ≠ 0
ELRSPREQ	Integer	User function element response flag = 1 Element response required = 0 Element response not required

Created By: Module MAKEST

Notes:

1. This relation is built from the CQDMEM1, associated P-type and the basic grid point data. It contains one tuple for each quadrilateral membrane element in the problem.
2. A nonzero SCON flag denotes that the element is affected by a stress constraint.
3. A nonzero DESIGN flag denotes that the element is affected by a design variable.
4. LAYRNUM is zero for noncomposite elements.

Entity: **QEE**

Entity Type: Matrix

Description: A partition of the GENQ matrix that contains only elastic degrees of freedom. Used in the nuclear blast response analysis.

Matrix Form: Real square matrix with the number of rows and columns equal to the number of elastic modes retained for the blast analysis.

Created By: MAPOL

Entity:	**QHHL**
Entity Type:	Matrix
Description:	Matrix list of generalized unsteady aerodynamic coefficients.
Matrix Form:	Complex rectangular matrix with one row for each retained mode shape and with the number of columns equal to the product of the number of retained mode shapes and the number of M-k pairs at which aerodynamics are required.
Created By:	Module QHHLGEN
Notes:	1. The matrix may be required in the flutter sensitivity analysis and is therefore subscripted by boundary condition.

Entity:	**QHJL**
Entity Type:	Matrix
Description:	Generalized aerodynamic data for the gust loads determination.
Matrix Form:	A variable-sized matrix list. Each Mach number and reduced frequency required in the gust analysis creates a matrix with one row for each retained mode panel and one volume for each aerodynamic panel.
Created By:	QHHLGEN
Notes:	1. See QJKL 2. The order of the matrices in the list is the order of M-k pairs in UNMK.

Entity:	**QJJL**
Entity Type:	Matrix
Description:	Matrix list of unsteady aerodynamic coefficients.
Matrix Form:	Complex rectangular matrix with one row for each aerodynamic panel and with the number of columns equal to the product of the number of aerodynamic panels and the number of M-k pairs at which aerodynamics are required.
Created By:	Module AMP
Notes:	1. This matrix list is generated only if there is a requirement for nuclear blast analysis. 2. The matrix components in QJJL are the inverse of the transpose of the matrix components of matrix AJJTL associated with BLAST analyses.

Entity:	**QKJL**
Entity Type:	Matrix
Description:	Aerodynamic interpolation list containing data required for gust analysis.
Matrix Form:	A variable-sized matrix list. There is an nk by nj matrix for each Mach number and reduced frequency required in the gust analysis.
Created By:	Module AMP
Notes:	1. The order of the matrices in the list is the order of M-k pairs in UNMK. 2. One matrix QKJ is generated for each M-k pair associated with gust analyses.

Entity:	**QKKL**
Entity Type:	Matrix
Description:	Matrix list of unsteady aerodynamic coefficients.
Matrix Form:	Complex rectangular matrix with one row for each aerodynamic degree of freedom and with the number of columns equal to the product of the number of aerodynamic degrees of freedom and the number of M-k pairs at which aerodynamics are required.
Created By:	Module AMP
Notes:	1. The matrix components of this list are generated from: $[QKK] = [SJK][AJJT]^{-T} [[D1JK] + ik[D2JK]]$ 2. One matrix QKK is generated for each M-k pair associated with flutter or gust analyses.

Entity:	**QRE**
Entity Type:	Matrix
Description:	A partition of the GENQ matrix.
Matrix Form:	Real rectangular matrix with the number of rows equal to the number of rigid body modes and the number of columns equal to the number of elastic modes retained for the blast analysis.
Created By:	MAPOL

Entity:	**QRR**
Entity Type:	Matrix
Description:	A partition of the GENQ matrix.
Matrix Form:	Real square matrix with the number of rows and columns equal to the number of rigid body modes in the blast analysis.
Created By:	MAPOL

Entity: **QUAD4EST**

Entity: Relation

Description: Contains the element summary data for the quadrilateral QUAD4 plate element.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0	Element identification number
PID	Integer > 0	Element property identification number
PTYPE	Text (8)	Element property type
LAYRNUM	Integer ≥ 0	Composite layer number
SIL1	Integer > 0	Internal grid point id 1
SIL2	Integer > 0	Internal grid point id 2
SIL3	Integer > 0	Internal grid point id 3
SIL4	Integer > 0	Internal grid point id 4
TRATIO1	Real > 0.0	Ratio of membrane thickness to element thickness for grid 1
TRATIO2	Real > 0.0	Ratio of membrane thickness to element thickness for grid 2
TRATIO3	Real > 0.0	Ratio of membrane thickness to element thickness for grid 3
TRATIO4	Real > 0.0	Ratio of membrane thickness to element thickness for grid 4
CID1	Integer ≥ 0	Coordinate system defining material axis
THETAM	Real	Material orientation angle
OFFST0	Real	Initial offset of the element reference plane from the plane of grid points.
MID1	Integer ≥ 0	Material identification number for membrane
THICK	Real > 0.0	Membrane thickness
MID2	Integer ≥ 0	Material identification number for bending
BENDSTIF	Real > 0.0	Bending stiffness parameter
MID3	Integer ≥ 0	Material identification number for transverse shear
TRNSVRS	Real > 0.0	Transverse shear thickness factor
NSM	Real > 0.0	nonstructural man
FZ1	Real > 0.0	Fiber distance for stress computation
FZ2	Real > 0.0	Fiber distance for stress computation
MID4	Integer ≥ 0	Material identification number for membrane-bending coupling
CIDS	Integer ≥ 0	Coordinate system defining stress output coordinate system

NAME	TYPE/KEY	DESCRIPTION
THETAS	Real	Stress output orientation angle
COORD1	Integer	External coord system for SIL1
X1, Y1, Z1	Real	Basic coordinates of SIL1
COORD2	Integer ≥ 0	External coord system for SIL2
X2, Y2, Z2	Real	Basic coordinates of SIL2
COORD3	Integer ≥ 0	External coord system for SIL3
X3, Y3, Z3	Real	Basic coordinates of SIL3
COORD4	Integer ≥ 0	External coord system for SIL4
X4, Y4, Z4	Real	Basic coordinates of SIL4
SCON	Integer	Stress constraint flag
DESIGN	Integer	Design flag
STHRM	Real Array(3)	Thermal stress terms for the constrained element
STHRMA	Real Array(3)	Thermal strain terms for the constrained element
TREFPT	Integer ≥ 0	Pointer to the TREF entity for thermal loads/stress evaluation of the designed element
NLFLAG	Integer ≥ 0	Nonlinear design variable flag = 0 Linear design variable = 1 Nonzero, nonstructural mass (NSM ≠ 0) = 2 Nonlinear design stiffness = 3 Nonlinear design stiffness and NSM ≠ 0 = 4 Nonlinear design stiffness and design mass = 5 Nonlinear design stiffness and mass and NSM ≠ 0
CMPFLG	Integer ≥ 0	Composite element type flag = 0 Noncomposite element (1 entry for layer "0") = 1 Nondesigned and nonconstrained composite element (1 entry for layer "0" for equiv. single layer) = 2 Nondesigned and constrained composite element (1 entry for layer "0" for equiv. single layer and *nlayer* entries at each layer) = 3 Designed membrane composite element entries at each layer = 4 Designed bending composite element (1st entry for equiv. single layer and entries at each layer)
ELTHK0	Real > 0.0	Initial user input membrane thickness

NAME	TYPE/KEY	DESCRIPTION
ELRSPREQ	Integer	User function element response flag = 1 Element response required = 0 Element response not required
LAYERPOS	Integer	This layer's position among all composite layers

Created By: Module MAKEST

Notes:

1. This relation is built from the CQUAD4, associated P-type and the basic grid point data. It contains one tuple for each isoparametric QUAD4 element in the problem.
2. The number of EST entries for a composite element could be equal to the number of layers, or to the number of layers + one, depending on the composite type. (See preceeding CMPFLG description)

Entity: **RANDPS**

Entity Type: Relation

Description: Contains the definition of load set power spectral density factors for use in Random analysis having the frequency dependent form.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Random analysis set identification number
EXCITID	Integer > 0	Subcase identification number of excited load set
APPLYID	Integer > 0	Subcase identification number of applied load set
X, Y	Real	Components of complex number
TABRNDID	Integer ≥ 0	Identification number of a TABRNDi entry

Created By: Module IFP

Entity: **RBAR**

Entity Type: Relation

Description: Contains the definition of a rigid bar element with six degrees of freedom at each end.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	MPC identification number
EID	Integer > 0	Rigid bar element identification number
GA,GB	Integer > 0	Grid point identification numbers of connection points.
CNA,CNB	Integer	Independent DOF in the global coordinate system for the elements at grid point GA and GB.
CMA,CMB	Integer	Dependent DOF in the global coordinate system assigned by the element at grid point GA and GB

Created By: Module IFP

Entity: **RBE1**

Entity Type: Relation

Description: Contains the definition of a rigid body connected to an arbitrary number of grid points.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	MPC identification number
EID	Integer > 0	Rigid body element identification number
GXI	Integer > 0	Grid point identification numbers at which dependent/independent DOF are assigned
CXI	Integer > 0	Component numbers of dependent/independent DOF in the global coordinate system at grid points GXi
UMFLAG	Text (4)	Character string indicating the start of the list of dependent degrees-of-freedom

Created By: Module IFP

Entity: **RBE2**

Entity Type: Relation

Description: Contains the definition of a rigid body whose independent degrees-of-freedom are specified at a single grid point and whose dependent degrees-of-freedom are specified at an arbitrary number of grid points.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	MPC identification number
EID	Integer > 0	Rigid body element identification number
GN	Integer > 0	Grid point identification number at which all six independent DOF are assigned
CM	Integer	Component numbers of dependent degrees-of-freedom in the global coordinate system assigned by the element at grid points GMi
GMI	Integer > 0	Grid point identification number at which dependent DOF are assigned

Created By: Module IFP

Entity: **RBE3**

Entity Type: Relation

Description: Contains the definition of the motion of a reference grid point as the weighted average of motions at a set of other grid points.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	MPC identification number
EID	Integer > 0	Rigid body element identification number
REFGRID	Integer > 0	Reference grid point identification number
REFC	Integer	Component numbers of DOF in the global coordinate system that will be computed at REFGRID
QI	Integer	Integer field of either grid point ID or component number
QR	Real	Real field of weighting factor
UMFLAG	Text (4)	Character string indicating the start of the list of dependent DOF

Created By: Module IFP

Entity: **RGCFRESP**

Entity Type: Relation

Description: Contains the user function requested steady aeroelastic rigid stability coefficient response values.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
VALUE	Real	Response value

Created By: Module SAERO

Entity: **RHS**

Entity Type: Subscripted Matrix

Description: A matrix used in the analysis of free-free structures that corresponds to load vectors applied to the supported degrees of freedom.

Matrix Form: The number of rows is equal to the number of degrees of freedom in the r-set while the number of columns varies by the type of analysis being performed.

Created By: MAPOL

Notes:

1. For an inertia relief analysis, RHS is equal to PR plus the transpose of D times PLBAR.
2. For a static aeroelastic analysis, RHS is equal to P2 minus K21 times PAR.
3. Since RHS may be needed in the sensitivity analysis, it is subscripted by boundary condition number.

Entity: **RLOAD1**

Entity Type: Relation

Description: Contains information on frequency dependent loads as defined in the RLOAD1 bulk data entry.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer > 0	Set identification number
ILAG	Integer	Identification number for DLAGS
ITC	Integer	Identification number for TABLEDi(C)
ITD	Integer	Identification number for TABLEDi(D)

Created By: Module IFP

Notes:

1. The relation is used in FRLGA to generate dynamic loads.

Entity: **RLOAD2**

Entity Type: Relation

Description: Contains information to define frequency dependent dynamic loads in a form specified in the RLOAD2 bulk data entry.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer > 0	Set identification number
ILAG	Integer	Identification number for DLAGS
TB	Integer	Identification number for TABLEDi(B)
TP	Integer	Identification number for TABLEDi(P)

Created By: Module IFP

Notes:

1. The relation is used in FRLGA to generate dynamic loads.

Entity: **RODEST**

Entity Type: Relation

Description: Contains the element summary data for the ROD element.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
PID	Integer > 0	Element property identification number
SIL1	Integer > 0	Internal id of grid at end A
SIL2	Integer > 0	Internal id of grid at end B
MID1	Integer > 0	Material id of a MAT1 tuple
AREA	Real ≥ 0.0	Element cross-sectional area
J	Real ≥ 0.0	Torsional constant
C	Real	Stress recovery coefficient
NSM	Real ≥ 0.0	Element nonstructural mass
COORD1	Integer ≥ 0	External coordinate system id for displacements at end A
X1, Y1, Z1	Real	Basic coordinates at end A
COORD2	Integer ≥ 0	External coordinate system id for displacements at end B
X2, Y2, Z2	Real	Basic coordinates at end B
SCON	Integer	Stress constraint flag
DESIGN	Integer	Design flag
STHRM	Real	Thermal stress term for the constrained element
STHRMA	Real	Thermal strain term for the constrained element
TREFPT	Integer ≥ 0	Pointer to TREF entity for thermal stress evaluation and thermal loads evaluation
NLFLAG	Integer ≥ 0	Nonlinear design variable flag = 0 linear design variable = 1 NSM ≠ 0
ELRSPREQ	Integer	User function element response flag = 1 Element response required = 0 Element response not required

Created By: Module MAKEST

Notes:

1. This relation is built from the CONROD, CROD, PROD and basic grid point relations. It contains one tuple for each ROD element in the problem.
2. A nonzero SCON flag denotes that the element is affected by a stress constraint.
3. A nonzero DESIGN flag denotes that the element is affected by a design variable.

Entity: **RROD**

Entity Type: Relation

Description: Contains the definition of a pin-ended rod that is rigid in extension.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	MPC identification number
EID	Integer > 0	Rigid rod element identification number
GA,GB	Integer > 0	Grid point identification numbers of connection points.
CMA,CMB	Integer	Component number of one dependent DOF in the global coordinate system assigned by the element at either grid point GA or GB

Created By: Module IFP

Entity: **R21**

Entity Type: Subscripted Matrix

Description: Intermediate matrix formed in the solution of structures that contain unrestrained degrees of freedom.

Matrix Form: Real rectangular matrix with one row for each r-set degree of freedom and one column for each a-set degree of freedom.

Created By: MAPOL

Notes:

1. R21 is the transpose of IFR.

Entity:	**R22**
Entity Type:	Matrix
Description:	Intermediate matrix formed in the solution of structures that contain unrestrained degrees of freedom.
Matrix Form:	Real square matrix with one row and column for each r-set degree of freedom.
Created By:	MAPOL
Notes:	1. R22 is created from: $[R22] = [D]^T[MLR] + [MRRBAR]$

Entity:	**R31**
Entity Type:	Subscripted Matrix
Description:	Intermediate matrix formed in the solution of static aeroelastic response.
Matrix Form:	Real rectangular matrix with one row and column for each r-set degree of freedom and one column for each l-set degree of freedom.
Created By:	MAPOL
Notes:	1. R31 is only computed for the steady aeroelastic analysis. 2. R31 is created from: $[R31] = [D]^T[KALL] + [KARL]$

Entity:	**R32**
Entity Type:	Subscripted Matrix
Description:	Intermediate matrix formed in the solution of static aeroelastic response.
Matrix Form:	Real square matrix with one row and column for each r-set degree of freedom.
Created By:	MAPOL
Notes:	1. R32 is only computed for the steady aeroelastic analysis. 1. R32 is created from: $[R32] = [D]^T[KALR] + [KARR]$

Entity: **SAVE**

Entity Type: Relation

Description: Contains a list of data base entities whose contents are to be saved rather then purged through the UTPURG utility.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
ENTNAM	Text (8)	The name of a database entity.

Created By: Module IFP

Notes:

1. An entity named in this relation will not have its contents purged by the UTPURG utility.

Entity: **SEQGP**

Entity Type: Relation

Description: Contains the user selected resequencing requests for the grid and scalar points of the structural modes.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EXTID	Integer > 0	Grid point identification number
SEQNUM	Text (8)	Sequenced identification number (see beolw)

Created By: Module IFP

Notes:

1. EXTID is any grid or scalar point identification number which is to be reidentified for sequencing purposes. The sequence number is a special number which may have any of the following forms where X is a decimal integer digit — XXXX.X.X.X, XXXX.X.X, XXXX.X, or XXXX where any of the leading X's may be omitted. This string contains no imbedded blanks. The leading character will not be a decimal point.

Entity: **SET1**

Entity Type: Relation

Description: Contains a list of structural grid points to be used in splining loads from aerodynamic points to structural points and modes to be omitted from flutter analyses.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
GRIDI	Integer > 0	Structural grid point id

Created By: Module IFP

Notes:

1. This relation contains one tuple for each grid point in each set.

Entity: **SET2**

Entity Type: Relation

Description: Contains the definition of a set of structural grid points in terms of aerodynamic elements. The set will be used to spline aero loads to the structure.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
SP1	Real	Lower spanwise division
SP2	Real	Upper spanwise division
CH1	Real	Lower chordwise division
CH2	Real	Upper chordwise division
ZMAX	Real	Z-coordinate of upper surface
ZMIN	Real	Z-coordinate of lower surface

Created By: Module IFP

Notes:

1. Tuples of this relation are referenced by the GRDSETID attribute of the SPLINE1 and SPLINE2 relations.

Entity: **SHAPE**

Entity Type: Relation

Description: Contains the element identification numbers and weighting factors specified on the SHAPE Bulk Data entry.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SHAPEID	Integer	Shape identification number
ETYPE1	Text (8)	Element type
EID1	Integer	Element identification number
PREF	Real	Design variable linking factor

Created By: Module IFP

Notes:

1. Allowable ETYPE1 entries are:
   - CROD, CONROD
   - CSHEAR
   - CQDMEM1
   - CQUAD4
   - CTRIA3
   - CTRMEM
   - CMASS1, CMASS2
   - CBAR
   - CONM2
   - CELAS1, CELAS2

Entity: **SHAPEM**

Entity Type: Relation

Description: Contains the element identification numbers and weighting factors specified on the SHAPEM Bulk Data entry.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SHAPEID	Integer	Shape identification number
ETYPE1	Text (8)	Element type
EID1	Integer	Element identification number
DVSYMBL	Text(8)	Designed dimension symbol
PREF	Real	Design variable linking factor

Created By: Module IFP

Notes:

1. Allowable ETYPE1 entries are:
   - CROD, CONROD
   - CSHEAR
   - CQDMEM1
   - CQUAD4
   - CTRIA3
   - CTRMEM
   - CMASS1, CMASS2
   - CBAR
   - CONM2
   - CELAS1, CELAS2
2. Allowable DVSYMBL are:
   - D1, D2, D3, D4, D5, D6, D7, D8, D9, D10 for BAR element cross section dimensions;
   - A for element cross section area;
   - T for element thickness;
   - M for element mass;
   - K for element thickness.

Entity: **SHEAREST**

Entity Type: Relation

Description: Contains the element summary data for the shear panel.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
PID	Integer > 0	Element property identification number
SIL1	Integer > 0	Internal grid point id
SIL2	Integer > 0	
SIL3	Integer > 0	
SIL4	Integer > 0	
MID1	Integer > 0	Material id of a MAT1 tuple
THICK	Real > 0.0	Element thickness
NSM	Real ≥ 0.0	Element nonstructural mass
COORD1	Integer ≥ 0	External coordinate system id for displacements at SIL1
X1, Y1, Z1	Real	Basic coordinates of SIL1
COORD2	Integer ≥ 0	External coordinate system id for displacements at SIL2
X2, Y2, Z2	Real	Basic coordinates of SIL2
COORD3	Integer ≥ 0	External coordinate system id for displacements at SIL3
X3, Y3, Z3	Real	Basic coordinates of SIL3
COORD4	Integer ≥ 0	External coordinate system id for displacements at SIL4
X4, Y4, Z4	Real	Basic coordinates of SIL4
SCON	Integer	Stress constrain flag
DESIGN	Integer	Design flag
NLFLAG	Integer ≥ 0	Nonlinear design variable flag = 0 linear design variable = 1 NSM ≠ 0
ELRSPREQ	Integer	User function element response flag = 1 Element response required = 0 Element response not required

Created By: Module MAKEST

Notes:

1. This relation is built from the CSHEAR, associated P-type and the basic grid point relations. It contains one tuple for each shear panel in the problem.
2. A nonzero SCON flag denotes that the element is affected by a stress constraint.
3. A nonzero DESIGN flag denotes that the element is affected by a design variable.

Entity: **SHPGEN**

Entity Type: Relation

Description: Contains the design variable identification numbers, the list of associated elements and the shape to be generated by the Shape Generation Utility as specified on the SHPGEN Bulk Data entry.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SHAPEID	Integer > 0	Design variable shape function identification number
ELMLID	Integer > 0	ELEMLIST set identification number for associated elements
SHAPE	Text(4)	Desired shape
X0	Real	X-coordinate in the basic system of the new origin for shape generation
Y0	Real	Y-coordinate in the basic system of the new origin for shape generation
Z0	Real	Z-coordinate in the basic system of the new origin for shape generation

Created By: Module IFP

Entity: **SKJ**

Entity Type: Matrix

Description: Unsteady aerodynamic integration matrix list that translates pressures into forces and moments.

Matrix Form: Real rectangular matrix with one row for each aerodynamic degree of freedom and one column for each aerodynamic panel for each M-k pair.

Created By: Module UNSTEADY

Entity: **SLPMOD**

Entity Type: Matrix

Description: Intermediate matrix in the nuclear blast response calculation to transform elastic eigenvectors into slopes at aerodynamic panel control points.

Matrix Form: Real rectangular matrix with one row for each aerodynamic panel and one column for each elastic eigenvector.

Created By: MAPOL

Entity: **SMAT**

Entity Type: Matrix

Description: Contains the linear portion of the sensitivity of the stress and strain in the element's coordinate system to the global displacements.

Matrix Form: A variable-sized double precision matrix having one column for every stress/strain term in each linearly designed element that is constrained by a stress/strain constraint tuple or its stress/strain responses are required by user functions, and one row for every structural degree of freedom. The columns are stored in the order the elements are processed in EMG. Relation SMATCOL has matrix column information for each related elements.

Created By: Module EMG

Notes:

1. This matrix is not built if no linearly designed elements' stresses or strains are constrained.
2. SMAT and NLSMAT are used by SCEVAL module for constraint evaluation and MAKDFU for sensitivity evaluation.

Entity: **SMATCOL**

Entity Type: Relation

Description: Contains the matrix [SMAT] column information for all realted elements.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0	Element identification number
ETYPE	Text (8)	Element type
LAYERNUM	Integer ≥ 0	Composite layer number
STRECOL	Integer > 0	The first column number in [SMAT] for this element/layer stress
NCOLSTRE	Integer > 0	The number of columns in [SMAT] for this element/layer stress
STRACOL	Integer > 0	The first column number in [SMAT] for this element/layer strain
NCOLSTRA	Integer > 0	The number of columns in [SMAT] for this element/layer strain

Created By: Module EMG

Entity: **SMPLOD**

Entity Type: Unstructured

Description: Simple load vector information.

Entity Structure: Record 1 contains three integers defining the number of (1) simple external loads, NEXTLD, (2) gravity loads, NGRAV, and (3) thermal loads, NTHERM, followed by a list of load identification numbers for each of the three groups in sorted order within each group.

The second through NEXTLD + 1 records contain the external loads.

Created By: Module LODGEN

Entity: **SPC**

Entity Type: Relation

Description: Contains the definition of the single point constraints and enforced displacements as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
GRID1	Integer > 0	Grid or scalar point id
COMPNTS1	Integer ≥ 0	Components of GRID1 that are constrained
ENFDISP	Real	The value of the enforced displacement at all coordinates specified by COMPNTS1

Created By: Module IFP

Notes:

1. This relation is used by the MKUSET module to build the single-point constraint set.

Entity: **SPC1**

Entity Type: Relation

Description: Contains the definition of the single point constraints as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
COMPNTS	Integer > 0	Components to be constrained
GRIDI	Integer > 0	Grid point id defining the constrained components

Created By: Module IFP

Notes:

1. This relation is used by the MKUSET module to build the single-point constraint set.
2. This relation will contain one tuple for each grid point specified in each unique set of COMPNTS; for example,

   COMPNTS = 236, grids 5,6, 8 and 9

   and

   COMPNTS = 134, grids 10, 20

   will result in 6 tuples.

Entity: **SPCADD**

Entity Type: Relation

Description: Contains the definition as input from the Bulk Data file of a single-point constraint sets as a union of SPC and/or of SPC1 sets.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
SPCSETID	Integer > 0	Set id of a SPC or SPC1 tuple

Created By: Module IFP

Notes:

1. This relation is used by the MKUSET module to build the single-point constraint set.

Entity: **SPLINE1**

Entity Type: Relation

Description: Contains the definition of surface splines used for interpolating out-of-plane motion in aeroelastic analysis.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
CP	Integer ≥ 0	Coordinate system defining the plane of the spline
CAEROID	Integer > 0	Aero element id
BOX1	Integer > 0	First aero box to use the spline
BOX2	Integer > 0	Last aero box to use the spline
GRDSETID	Integer > 0	Set id of a SETi tuple defining the structural grids
FLEX	Real	Linear attachment flexibility

Created By: Module IFP

Notes:

1. Aerodynamic boxes are numbered sequentially in chordwise strips.

Entity: **SPLINE2**

Entity Type: Relation

Description: Contains the definition of a beam spline for interpolating panels and bodies for steady and unsteady aeroelastic analysis.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0	Element identification number
MACROID	Integer > 0	The identification of the aerodynamic macroelement to be splined
BOX1,BOX2	Integer > 0	The identification numbers of the first and last boxes on the macroelement to be interpolated using this spline
GRDSETID	Integer > 0	The identification of a SETi entry which lists the structural grid points to which the spline is attached
FLEX	Real ≥ 0.0	Linear attachment flexibility
DTOR	Real ≥ 0.0	Torsional flexibility
CID	Integer	Rectangular coordinate system which defines the y-axis of the spline
DTHX,DTHY	Real	Rotational attachment flexibility about the x-axis and y-axis

Created By: Module IFP

Entity: **SPOINT**

Entity Type: Relation

Description: Contains the identification numbers of those points to be used as scalar points. Input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EXTID	Integer > 0	External point identification

Created By: Module IFP

Entity: **STABCF**

Entity Type: Relation

Description: Rigid body stability coefficients.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
MACHINDX	Integer	Mach number index of associated AIRFRC
PARM	String(8)	Character string identifying the configuration parameter.
SYMFLG	Integer	Symmetry flag for the parameter.
PARMVAL	Real	Parameter value used to generate the "unit" forces.
CL	Real	Lift coefficient
CD	Real	Drag coefficient
CS	Real	Sideforce coefficient
CMX	Real	Rolling moment coefficient
CMY	Real	Pitching moment coefficient
CMZ	Real	Yawing moment coefficient

Created By: Module STEADY

Notes:

1. The SYMFLG values are:
   = 1 Symmetric
   = -1 Antisymmetric
2. PARM identifies the physical variable whose perturbation generated the rigid coefficients. There are six accelerations and six configuration parameters whose names are reserved that have special meaning. Additional PARM values come from the set of all AESURF control surfaces defined and the PARM attribute contains the user supplied label. For a given MINDEX value, the AIRFRC matrix has one column (which may contain only zeros) for each entry in STABCF in the order of the STABCF relation. The PARM field is then:

PARM	VARIABLE
NX	Rigid body acceleration in drag/thrust direction (Produces no forces, but included for completeness to allow modification of AIRFRC columns to include nonzero terms)
NY	Rigid body acceleration in side force direction (Produces no forces, but included for completeness to allow modification of AIRFRC columns to include nonzero terms)

PARM	VARIABLE
NZ	Rigid body acceleration in plunge direction (Produces no forces, but included for completeness to allow modification of AIRFRC columns to include nonzero terms)
PACCEL	Rigid body acceleration about the roll axis. (Produces no forces, but included for completeness to allow modification of AIRFRC columns to include nonzero terms)
QACCEL	Rigid body acceleration about the pitch axis. (Produces no forces, but included for completeness to allow modification of AIRFRC columns to include nonzero terms)
RACCEL	Rigid body acceleration about the yaw axis. (Produces no forces, but included for completeness to allow modification of AIRFRC columns to include nonzero terms)
THKCAM	Forces arising from the effects of only thickness and camber with all other configuration parameters set to zero.
ALPHA	Forces arising due to unit angle of attack.
BETA	Forces arising due to a unit yaw angle.
PRATE	Forces arising due to a unit roll rate.
QRATE	Forces arising due to a unit pitch rate.
RRATE	Forces arising due to a unit yaw rate
surface	Forces arising due to the unit deflection of the AESURF control surface named in the PARM field.

Entity: **SUPORT**

Entity Type: Relation

Description: Contains the definition of the set of points, as input from the Bulk Data file, at which the user desires determinate reactions to be applied to a free body.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
GRID1	Integer > 0	Grid or scalar point identification
COMPNTS1	Integer ≥ 0	Components of GRID1

Created By: Module IFP

Notes:

1. This relation will be used by the MKUSET relation to build the support set.

Entity: **TABDMP1**

Entity Type: Relation

Description: Contains modal structural damping tables for use in flutter analysis as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Set identification number
TYPE	Text (4)	Damping type
FI	Real > 0.0	Frequency value
FBCD	Text (4)	A character attribute to process SKIP requests
GI	Real	Damping value
GBCD	Text (4)	A character attribute to process SKIP requests

Created By: Module IFP

Entity: **TABLED1**

Entity Type: Relation

Description: Contains tabular function data for generating dynamic loads as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
TID	Integer	Table identification number
XI	Real	Time (or frequency) for the tuple
YI	Real	Response for this tuple
STRXF	Text (4)	A character attribute to process skip requests
STRYI	Text (4)	A character attribute to process skip requests

Created By: IFP

Notes:

1. The relation is used in subroutine PRTAB1 to define time or frequency dependent load.

Entity: **TELM**

Entity Type: Unstructured

Description: Contains the element thermal loads partitions for nondesigned and linearly designed elements if any thermal loads have been defined in the model.

Entity Structure:

Record:

i. Each record contains the thermal loads partitions for each nondesigned and linear designed element in the model if any thermal loads have been defined in the model.

Created By: Module EMG

Notes:

1. This entity contains one record for each partition of each element thermal loads matrix. A partition is that portion of the matrix connected to one pivot sil.
2. Refer to the DVCT relation documentation for further details.
3. The TELM terms are stored in the same precision as the PG matrix.
4. The records related to nonlinear designed elements are empty.
5. TELM and DTELM are used to generate all of the element thermal load sensitivity matrix partitions.
6. TELM and TELMD are used to generate all of the element thermal load matrix partitions.

Entity: **TELMD**

Entity Type: Unstructured

Description: Contains the element thermal loads partitions for nonlinear design elements if any thermal loads have been defined in the model.

Entity Structure:

Record:

i. Each record contains the thermal loads partitions for each nonlinear design element in the model if any thermal loads have been defined in the model.

Created By: Module NLEMG

Notes:

1. This entity contains one record for each partition of each nonlinear design element thermal loads matrix. A partition is that portion of the matrix connected to one pivot sil.
2. Refer to the DVCTD relation documentation for further details.
3. The TELMD terms are stored in the same precision as the PG matrix.
4. TELMD and TELM are used to generate all of the element thermal load matrix partitions.

Entity: **TEMP**

Entity Type: Relation

Description: Contains the grid point temperatures as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer	The set identification number
GRID1	Integer	The grid point id
TEMPVAL	Real	The value of temperature assigned to GRID1

Created By: Module IFP

Entity: **TEMPD**

Entity Type: Relation

Description: Contains the default grid point temperature as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer	The set identification
TEMPDVAL	Real	The default grid point temperature for the set SETID

Created By: Module IFP

Entity: **TF**

Entity Type: Relation

Description: Contains the definition of transfer functions as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer	Set identification number
GD	Integer	Grid, scalar or extra point id
CD	Integer	Component number of grid point GD
B0	Real	Zeroth order coefficient
B1	Real	First order coefficient
B2	Real	Second order coefficient
GI	Integer	Grid, scalar or extra point id
CI	Integer	Component number of grid point GI
A0I	Real	Zeroth order coefficient
A1I	Real	First order coefficient
A2I	Real	Second order coefficient

Created By: Module IFP

Entity: **TFDATA**

Entity Type: Unstructured

Description: Contains the collected transfer function data for all transfer function sets defined.

Entity Structure:

Record:

1. A list of all set identification numbers in sorted order
i. Contains the transfer function for the (i-1)th transfer function set. Each record has the following form:

WORD	VARIABLE	DESCRIPTION
1	SID	Set identification for the (i-1)th transfer function set
j	COL	Internal number of the matrix column affected by the transfer function
j+1	NROW	Number of terms defined in the column COL
j+2 to j+1+4*NROW		For each term in the column four words are stored: 1) Internal number of the matrix row 2) 0th order coefficient 3) 1st order coefficient 4) 2nd order coefficient in sorted row order

Created By: PFBULK

Notes:

1. This entity is used in DMA to assemble dynamic matrices.
2. The j index runs from 1 to NCOL for each column in the matrix that is affected by the transfer function terms in sorted column order.

Entity: **TFIXED**

Entity Type: Relation

Description: Contains the layer thicknesses of undesigned layers of designed composite elements.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0	Element identification number
ETYPE	Text (8)	Element type. One of the following: QDMEM1 QUAD4 TRMEM TRIA3
LAYRNUM	Integer > 0	Layer number
T	Real > 0.0	Thickness

Created By: MAKEST

Notes:

1. These thicknesses are used in the evalation of thickness constraints and composite laminate constraints

Entity: **THKDRVA**

Entity Type: Relation

Description: Contains the user function requested element thickness sensitivity information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
MXCOL	Integer > 0	Matrix column number in [MXTHKDVA] for the sensitivity values

Created By: Module MAKDFV

Entity: **THKRESP**

Entity Type: Relation

Description: Contains the user function requested element thickness values.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
VALUE	Real	Response value

Created By: Module TCEVAL.

Entity: **TIMELIST**

Entity Type: Relation

Description: Contains the list of times for which outputs are requested as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer	Set identification number
TIME	Real	Time value in consistent units

Created By: Module IFP

Entity: **TLOAD1**

Entity Type: Relation

Description: Contains information on time dependent loads as defined on the TLOAD1 bulk data entry.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer > 0, key	Set identification number
IDEL	Integer > 0	ID of the DLAGS set
TABL1	Integer	ID of the TABLED1 set

Created By: Module IFP

Notes:

1. The relation is used in module OFPLOAD to generate dynamic loads.

Entity: **TLOAD2**

Entity Type: Relation

Description: Contains information on time dependent loads as defined by the TLOAD2 bulk data entry.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer > 0, key	Set identification number
IDEL	Integer > 0	ID of the DLAGS set
T1	Real ≥ 0.0	Time constant
T2	Real > T	Time constant
FREQ	Real ≥ 0.0	Frequency parameter
PHASE	Real	Phase parameter
CTEXP	Real	Exponential coefficient
GROWTH	Real	Growth coefficient

Created By: Module IFP

Notes:

1. The relation is used in module OFPLOAD to generate dynamic loads.

Entity:	**TMN**
Entity Type:	Subscripted Matrix
Description:	Contains the rigid constraint matrix relating the displacements at dependent degrees of freedom to those at the independent degrees of freedom.
Matrix Form:	A variable-sized single precision matrix having one row for each dependent degree of freedom and one column for each independent degree of freedom. The rigid constraint matrix is built from MPC and rigid elements such that: $[u_m] = [TMN]\ [u_n]$
Created By:	Module MKUSET
Notes:	1. The dimension of this subscripted matrix must be large enough for all optimization and analysis boundary conditions. 2. If no multipoint constraints are defined, this matrix will have no columns.

Entity:	**TMP1**
Entity Type:	Matrix
Description:	A scratch matrix used at various points in the MAPOL sequence for intermediate calculation.
Matrix Form:	Application dependent.
Created By:	MAPOL

Entity:	**TMP2**
Entity Type:	Matrix
Description:	A scratch matrix used at various points in the MAPOL sequence for intermediate calculation.
Matrix Form:	Application dependent.
Created By:	MAPOL

Entity: **TREF**

Entity Type: Unstructured

Description: Contains the element reference temperature for each linear design element in the model.

Entity Structure:

Record:

1. Contains the reference temperature for each linear design element in the model. The temperatures are stored in the order the elements are processed.

Created By: Module EMG

Notes:

1. Elements are processed alphabetically by element type and numerically within each element type.
2. Entity is only created if TEMP or TEMPD bulk data entries exist.
3. The TREFPT attribute on the XXXEST relations points to the position in TREF for the associated reference temperature.

Entity: **TREFD**

Entity Type: Unstructured

Description: Contains the element reference temperature for each nonlinear design element in the model.

Entity Structure:

Record:

1. Contains the reference temperature for each nonlinear design element in the model. The temperatures are stored in the order the elements are processed.

Created By: Module NLEMG

Notes:

1. Elements are processed alphabetically by element type and numerically within each element type.
2. Entity is only created if TEMP or TEMPD bulk data entries exist.
3. The TREFPT attribute on the XXXEST relations points to the position in TREF for the associated reference temperature.

Entity: **TRIM**

Entity Type: Relation

Description: Contains the specified conditions for steady aeroelastic trim or nonplanar steady aerodynamic analysis as input from Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Trim set identification number
MACH	Real > 0.0	Mach number
QDP	Real > 0.0	Dynamic pressure
TRMTYP	Text(8)	Type of trim desired
EFFID	Integer	Identification of CONEFFS bulk data entries which modify control surface effectiveness values
V0	Real	Velocity
LABELI	Text(8)	Label defining the aerodynamic trim parameters
FREEI	Text(4)	Character string FREE
FIXI	Real	Magnitude of the trim parameter

Created By: Module IFP and STEADY

Entity: **TRIMDATA**

Entity Type: Relation

Description: Contains the TRIM Bulk Data and related boundary condition, subcase and subscript information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SETID	Integer > 0	Trim set identification number
MACH	Real > 0.0	Mach number
QDP	Real > 0.0	Dynamic pressure
TRMTYP	Text(8)	Type of trim desired
EFFID	Integer	Identification of CONEFFS bulk data entries which modify control surface effectiveness values
V0	Real	Velocity
LABELI	Text(8)	Label defining the aerodynamic trim parameters
FREEI	Text(4)	Character string FREE
FIXI	Real	Magnitude of the trim parameter
BCID	Integer	Boundary condition identification number
MACHINDX	Integer	Mach number index for the current subcase
SYMMFLAG	Integer	Symmetry option for the Mach Number
SUBSCRPT	Integer	Subscript counter
SUBCASID	Integer	Subcase identification number

Created By: Module TRIMCHEK

Entity: **TRIMDRVA**

Entity Type: Relation

Description: Contains the user function requested steady aeroelastic trim parameter response sensitivity information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
MXCOL	Integer > 0	Matrix column number in [MXTRMDVA] for the sensitivity values

Created By: Module AEROSENS

Entity: **TRIMRESP**

Entity Type: Relation

Description: Contains the user function requested steady aeroelastic trim parameter response values.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
VALUE	Real	Response value

Created By: Module SAERO

Entity: **TRIA3EST**

Entity: Relation

Description: Contains the element summary data for the triangular TRIA3 element.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0	Element identification number
PID	Integer > 0	Element property identification number
PTYPE	Text (8)	Element property type
LAYRNUM	Integer ≥ 0	Composite layer number
SIL1	Integer > 0	Internal grid point id 1
SIL2	Integer > 0	Internal grid point id 2
SIL3	Integer > 0	Internal grid point id 3
TRATIO1	Real > 0.0	Ratio of membrane thickness to element thickness for grid 1
TRATIO2	Real > 0.0	Ratio of membrane thickness to element thickness for grid 2
TRATIO3	Real > 0.0	Ratio of membrane thickness to element thickness for grid 3
CID1	Integer ≥ 0	Coordinate system defining material axis
THETAM	Real	Material orientation angle
OFFST0	Real	Initial offset of the element reference plane from the plane of grid points.
MID1	Integer ≥ 0	Material identification number for membrane
THICK	Real > 0.0	Membrane thickness
MID2	Integer ≥ 0	Material identification number for bending
BENDSTIF	Real > 0.0	Bending stiffness parameter
MID3	Integer ≥ 0	Material identification number for transverse shear
TRNSVRS	Real > 0.0	Transverse shear thickness factor
NSM	Real > 0.0	Nonstructural mass
FZ1	Real > 0.0	Fiber distance for stress computation
FZ2	Real > 0.0	Fiber distance for stress computation
MID4	Integer ≥ 0	Material identification number for membrane-bending coupling

NAME	TYPE/KEY	DESCRIPTION
CIDS	Integer ≥ 0	Coordinate system defining stress output coordinate system
THETAS	Real	Stress output orientation angle
COORD1	Integer	External coord system for SIL1
X1, Y1, Z1	Real	Basic coordinates of SIL1
COORD2	Integer ≥ 0	External coord system for SIL2
X2, Y2, Z2	Real	Basic coordinates of SIL2
COORD3	Integer ≥ 0	External coord system for SIL3
X3, Y3, Z3	Real	Basic coordinates of SIL3
SCON	Integer	Stress constraint flag
DESIGN	Integer	Design flag
STHRM	Real Array(3)	Thermal stress terms for the constrained element
STHRMA	Real Array(3)	Thermal strain terms for the constrained element
TREFPT	Integer ≥ 0	Pointer to the TREF entity for thermal loads/stress evaluation of the designed element
NLFLAG	Integer ≥ 0	Nonlinear design variable flag = 0 Linear design variable = 1 Nonzero, nonstructural mass (NSM ≠ 0) = 2 Nonlinear design stiffness = 3 Nonlinear design stiffness and NSM ≠ 0 = 4 Nonlinear design stiffness and design mass = 5 Nonlinear design stiffness and mass and NSM ≠ 0
ELRSPREQ	Integer	User function element response flag = 1 Element response required = 0 Element response not required
LAYERPOS	Integer	This layer's position among all composite layers

NAME	TYPE/KEY	DESCRIPTION
CMPFLG	Integer ≥ 0	Composite element type flag = 0 Noncomposite element (1 entry for layer "0") = 1 Nondesigned and nonconstrained composite element (1 entry for layer "0" for equiv. single layer) = 2 Nondesigned and constrained composite element (1 entry for layer "0" for equiv. single layer and *nlayer* entries at each layer) = 3 Designed membrane composite element entries at each layer = 4 Designed bending composite element (1st entry for equiv. single layer and entries at each layer)
ELTHK0	Real > 0.0	Initial user input membrane thickness

Created By: Module MAKEST

Notes:

1. This relation is built from the CTRIA3, associated P-type and the basic grid point data. It contains one tuple for each isoparametric TRIA3 element in the problem.
2. The number of EST entries for a composite element could be equal to the number of layers, or to the number of layers + one, depending on the composite type. (See preceeding CMPFLG description)

Entity: **TRMEMEST**

Entity Type: Relation

Description: Contains the element summary data for the constant strain triangular membrane element.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
EID	Integer > 0, key	Element identification number
PID	Integer > 0	Element property identification number
PTYPE	Text (8)	Element property type
LAYRNUM	Integer ≥ 0	Composite layer number
SIL1	Integer > 0	Internal grid point id
SIL2	Integer > 0	
SIL3	Integer > 0	
CID	Integer ≥ 0	Coordinate system defining material axis
THETA	Real	Material orientation angle for anisotropic material behavior
MID1	Integer > 0	Material id of MAT1 tuple
THICK	Real > 0.0	Element thickness
NSM	Real ≥ 0.0	Element nonstructural mass
COORD1	Integer ≥ 0	External coordinate system id for displacements at SIL1
X1, Y1, Z1	Real	External coordinate system id for displacements at SIL1
COORD2	Integer ≥ 0	External coordinate system id for displacements at SIL2
X2, Y2, Z2	Real	External coordinate system id for displacements at SIL2
COORD3	Integer ≥ 0	External coordinate system id for displacements at SIL3
X3, Y3, Z3	Real	External coordinate system id for displacements at SIL3
SCON	Integer	Stress constraint flag
DESIGN	Integer	Design flag
STHRM	Real Array (3)	Thermal stress terms for the constrained element

NAME	TYPE/KEY	DESCRIPTION
STHRMA	Real Array (3)	Thermal strain terms for the constrained element
TREFPT	Integer ≥ 0	Pointer to TREF entity used to evaluate thermal loads and thermal stresses
NLFLAG	Integer ≥ 0	Nonlinear design variable flag = 0 linear = 1 NSM ≠ 0
ELRSPREQ	Integer	User function element response flag = 1 Element response required = 0 Element response not required

Created By: Module MAKEST

Notes:

1. This relation is built from the CTRMEM, associated P-type and the basic grid point relations. It contains one tuple for each triangular membrane element in the problem.
2. A nonzero SCON flag denotes that the element is affected by a stress constraint.
3. A nonzero DESIGN flag denotes that the element is affected by a design variable.
4. LAYRNUM is zero for noncomposite elements.

Entity: **TSTEP**

Entity Type: Relation

Description: Contains time step information for the dynamic response as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer > 0	Time step identification number
NDTI	Integer ≥ 2	Number of time steps for this tuple
DELTAI	Real > 0.0	Time increment for this tuple
NOUTI	Integer	Skip factor for this tuple

Created By: Module IFP

Notes:

1. The response at every NOUTIth time step will be saved for output.

Entity: **UA**

Entity Type: Matrix

Description: Displacements in the a-set.

Matrix Form: A variable-size matrix having one row for each degree of freedom in the analysis set and one column for each load condition in the current boundary condition.

Created By: See Notes.

Notes:

1. This matrix is calculated using:

   [K11] [UA] = [P1]

   if there is inertia relief,

   [UA] = [K1112] [AR] + [PAR] [DELTA]

   for static aeroelasticity, and

   [KAA] [UA] = [PA]

   for static analysis without inertia relief.
2. See UG.

Entity: **UBLASTG**

Entity Type: Matrix

Description: Blast response quantities in the g-set.

Matrix Form: Real rectangular matrix with one row for each g-set degree of freedom and three columns (corresponding to displacement, velocity, and acceleration) for each time step at which transient calculations are retained.

Created By: MAPOL

Entity: **UBLASTI**

Entity Type: Matrix

Description: Blast response quantities in the i-set.

Matrix Form: Real rectangular matrix with one row for each retained mode and three columns (corresponding to displacement, velocity, and acceleration) for each time step at which transient calculations are retained.

Created By: BLASTDRV

Entity: **UDLOLY**

Entity Type: Unstructured

Description: Contains collected DLONLY information.

Record:

1. ID's of the NDIS DLONLY sets in sorted order. Contains data for the (i-1)th DLONLY set. The information on each of these records is:

WORD NO	VARIABLE	DESCRIPTION
j	LOAD	Load factor
j+1	ISIL	Internal ID of load component

Created By: Module PFBULK

Notes:

1. The number of words for the ith record is twice the number of load factors input for the associated set ID.

Entity: **UF**

Entity Type: Matrix

Description: Displacements in the f-set derived from UO and UA (see UG).

Entity:	**UFREQA**
Entity Type:	Matrix
Description:	Matrix of frequency response quantities in the a-set.
Matrix Form:	A complex rectangular matrix with one row for each a-set degree of freedom and three columns (corresponding to displacement, velocity, and acceleration) for each frequency at which frequency response output is required.
Created By:	Module DYNRSP or MAPOL
Notes:	1. If the direct method of frequency response is used, UFREQ is computed in module DYNRSP. If the modal method is used, UFREQA is recovered using UFREQI and PHIA.

Entity:	**UFREQE**
Entity Type:	Matrix
Description:	Matrix of frequency response quantities in the e-set.
Matrix Form:	A complex rectangular matrix with one row for each e-set degree of freedom and three columns (corresponding to displacement, velocity, and acceleration) for each frequency at which frequency response output is required.
Created By:	Module DYNRSP
Notes:	1. UFREQE is only computed in a frequency response analysis that includes extra points.

Entity:	**UFREQF**
Entity Type:	Matrix
Description:	Matrix of frequency response quantities in the f-set.
Matrix Form:	A complex rectangular matrix with one row for each f-set degree of freedom and three columns (corresponding to displacement, velocity, and acceleration) for each frequency at which frequency response output is required.
Created By:	MAPOL

Entity:	**UFREQG**
Entity Type:	Matrix
Description:	Matrix of frequency response quantities in the g-set.
Matrix Form:	A complex rectangular matrix with one row for each g-set degree of freedom and three columns (corresponding to displacement, velocity, and acceleration) for each frequency at which frequency response output is required.
Created By:	MAPOL

Entity: **UFREQI**

Entity Type: Matrix

Description: Matrix of frequency response quantities in the i-set.

Matrix Form: A complex rectangular matrix with one row for each i-set degree of freedom and three columns (corresponding to displacement, velocity, and acceleration) for each frequency at which frequency response output is required.

Created By: Module DYNRSP

Notes:

1. This matrix is only computed when the modal method of frequency response is invoked.

Entity: **UFREQN**

Entity Type: Matrix

Description: Matrix of frequency response quantities in the n-set.

Matrix Form: A complex rectangular matrix with one row for each n-set degree of freedom and three columns (corresponding to displacement, velocity, and acceleration) for each frequency at which frequency response output is required.

Created By: MAPOL

Entity: **UG**

Entity Type: Subscripted Matrix

Description: Displacements of the structural degrees of freedom in the g-set.

Matrix Form: A variable-sized matrix having one row for each structural degree of freedom and one column for each load condition in the boundary condition.

Created By: MAPOL

Notes:

1. The MAPOL sequence recovers this matrix in the following order (see the Theoretical Manual for the explicit form of this recovery):

$$\begin{bmatrix} \mathbf{UO} \\ \mathbf{UA} \end{bmatrix} \rightarrow \mathbf{UF}$$

$$\begin{bmatrix} \mathbf{YS} \\ \mathbf{UF} \end{bmatrix} \rightarrow \mathbf{UN}$$

$$\begin{bmatrix} \mathbf{UM} \\ \mathbf{UN} \end{bmatrix} \rightarrow \mathbf{UG}$$

Entity: **UGA**

Entity Type: Matrix

Description: "Active" displacements vectors for the current boundary condition.

Matrix Form: The matrix has one column for each active displacement vector and GSIZE rows.

Created By: MAPOL

Notes:

1. This matrix is obtained by partitioning UG using the PGA partitioning vector.

Entity: **UGTKAB**

Entity Type: Matrix

Description: A partition of the UGTKF matrix (see UGTKG).

Entity: **UGTKA**

Entity Type: Matrix

Description: Unsteady spline matrix in the a-set derived from UGTKF (see UGTKG).

Entity: **UGTKF**

Entity Type: Matrix

Description: Unsteady spline matrix in the f-set derived from UGTKN (see UGTKG).

Entity: **UGTKG**

Entity Type: Matrix

Description: Matrix containing the spline relations which relate the structural and unsteady aerodynamics models

Matrix Form: Real rectangular matrix with one row for each g-set degree of freedom and one column for each aerodynamic degree of freedom.

Created By: Module SPLINEU

Notes:

1. The MAPOL sequence supports the following partitions of the UGTKG matrix (see the Theoretical Manual for the exact formation of these matrices):

$$[\mathbf{UGTKG}] \rightarrow \begin{bmatrix} \varphi \\ \mathbf{UGTKN} \end{bmatrix}$$

$$[\mathbf{UGTKA}] = [\mathbf{UGTKAB}] + [\mathbf{GSUBO}]^{T}[\mathbf{UGTKO}]$$

Entity:	**UGTKN**
Entity Type:	Matrix
Description:	Unsteady spline matrix in the n-set derived from UGTKG (see UGTKG).

Entity:	**UGTKO**
Entity Type:	Matrix
Description:	Unsteady spline matrix in the o-set obtained as a partition of UGTKF (see UGTKG).

Entity:	**UKQ**
Entity Type:	Matrix
Description:	Upper triangular portion of the decomposed KEQE matrix.
Matrix Form:	Square real matrix having one row and column for each elastic mode retained in the nuclear blast response analysis.
Created by:	DECOMP
Notes:	1. KEQE is not symmetric. 2. This matrix is formed for use by the GFBS large matrix utility.

Entity:	**UM**
Entity Type:	Matrix
Description:	Displacements in the m-set derived from UN and TMN (see UG).

Entity:	**UN**
Entity Type:	Matrix
Description:	Displacements in the n-set derived from UF and YS (see UG).

Entity: **UNMK**

Entity Type: Unstructured

Description: Contains a global list of Mach number and reduced frequency pairs for which aerodynamic matrices were generated in the aerodynamic matrix lists.

Entity Structure:

RECORD	WORDS	DESCRIPTION
1	1-6	A one-word entry for each combination of symmetry options in the order noted containing the number of m-k pairs having the particular symmetry option
	7 Thru 6+4*nmk	Contains one four word entry for each aerodynamic matrix selected for generation by the MKAEROi entries of the following form: M, K, SYMXZ, SYMXY
2	1 Thru 2*BGRP	Contains the number of j degrees of freedom and the number of k degrees of freedom for each unsteady aerodynamic group

Created By: Module UNSTEADY

Notes:

1. Record 1 is sorted first by Mach number (M) and then by reduced frequency (k) within each M value for each combination of symmetry values. The symmetry options are treated in the following order:

ORDER	SYMXZ	SYMXY
1	-1	-1
2	-1	0
3	0	-1
4	0	0
5	1	-1
6	1	0

Entity: **UO**

Entity Type: Matrix

Description: Sensitivities of displacements in the o-set.

Matrix Form: A real rectangular matrix with one row for each o-set degree of freedom and one column for each active subcase times the number of design variables.

Created By: MAPOL

Notes:

1. For static aeroelastic analysis, UO is computed from:

   [UO] = [GASUBO] [DUAV] + [UOO]

2. For inertia relief, UO is computed from:

   [UO] = [GSUBO] [DUAV] + [UOO]

Entity:	**UOO**
Entity Type:	Matrix
Description:	Intermediate displacement sensitivities of the o-set.
Matrix Form:	A real rectangular matrix with one row for each o-set degree of freedom and one column for each active subcase times the number of design variables.
Created By:	GFBS

Notes:	1. UOO is computed from:
	`[KAOO] [UOO] = [ [DPOV] + [POARO] [DDELDV] ]`
	2. For inertia relief, UOO is computed from:
	`[KAOO] [UOO] = [ [DPOV] + [IFM] [DUAD] ]`

Entity:	**URDB**
Entity Type:	Matrix
Description:	Vector of aircraft rigid body accelerations that are computed for aircraft trim prior to the nuclear blast response calculation.
Matrix Form:	Real rectangular matrix having two rows (vertical and angular acceleration) and one column.
Created By:	BLASTRIM

Entity: **USET**

Entity Type: Unstructured

Description: Contains the bit masks defining the structural sets to which the degrees of freedom belong.

Entity Structure:

Record:

i Each record contains the boundary condition id as the first word followed by one word for each dependent set containing the number of DOF in each dependent set. These are followed by one word for each degree of freedom containing the bit masks defining the structural sets to which they belong.

Created By: Module MKUSET

Notes:

1. This entity contains one record for each boundary condition in the problem.
2. The bit masks are used to generate matrix partitioning vectors.
3. The 11th word of the INFO array for this entity contains the number of degrees of freedom in the structural model (g- set size).
4. The USET header has the following form:

WORD	DESCRIPTION
1	Boundary condition id
2	Number of m-set dofs
3	Number of s-set dofs
4	Number of o-set dofs
5	Number of r-set dofs

Entity: **UTRANA**

Entity Type: Matrix

Description: Matrix of transient response quantities in the a-set.

Matrix Form: Complex rectangular matrix with one row for each a-set degree of freedom and three columns (corresponding to displacement, velocity, and acceleration) for each time step at which transient response output is required.

Created By: Module DYNRSP or MAPOL

Notes:

1. If the direct method of transient response is used, UTRANA is computed in module DYNRSP. If the modal method is used, UTRANA is recovered using UTRANI and PHIA.

Entity:	**UTRANE**
Entity Type:	Matrix
Description:	Matrix of frequency response quantities in the e-set.
Matrix Form:	Complex rectangular matrix with one row for each e-set degree of freedom and three columns (corresponding to displacement, velocity, and acceleration) for each time step at which transient response output is required.
Created By:	Module DYNRSP

Entity:	**UTRANF**
Entity Type:	Matrix
Description:	Matrix of frequency response quantities in the f-set.
Matrix Form:	Complex rectangular matrix with one row for each f-set degree of freedom and three columns (corresponding to displacement, velocity, and acceleration) for each time step at which transient response output is required.
Created By:	MAPOL

Entity:	**UTRANG**
Entity Type:	Matrix
Description:	Matrix of frequency response quantities in the g-set.
Matrix Form:	Complex rectangular matrix with one row for each g-set degree of freedom and three columns (corresponding to displacement, velocity, and acceleration) for each frequency at which frequency response output is required.
Created By:	MAPOL

Entity:	**UTRANI**
Entity Type:	Matrix
Description:	Matrix of frequency response quantities in the i-set.
Matrix Form:	Complex rectangular matrix with one row for each i-set degree of freedom and three columns (corresponding to displacement, velocity, and acceleration) for each time step at which transient response output is required.
Created By:	Module DYNRSP

Notes:

1. This matrix is only computed when the modal method of transient response analysis is invoked.

Entity: **UTRANN**

Entity Type: Matrix

Description: Matrix of frequency response quantities in the n-set.

Matrix Form: Complex rectangular matrix with one row for each n-set degree of freedom and three columns (corresponding to displacement, velocity, and acceleration) for each time step at which transient response output is required.

Created By: MAPOL

Entity: **VELOLIST**

Entity Type: Relation

Description: Contains the list of velocity values as input from the Bulk Data file.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer	Set identification number
VELOCITY	Real > 0.0	Velocity value

Created By: Module IFP.

Entity: **VSDAMP**

Entity Type: Relation

Description: Contains the specification of parameters used to generate viscous damping terms in the dynamic matrices.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
SID	Integer	Set identification number
GVAL	Real	Damping value
OMEGA3	Real	Cyclic frequency

Created By: Module IFP

Notes:

1. If both GVAL and OMEGA3 are nonzero, equivalent structural damping is used to generate the BDD and/or BHH entities.
2. If only GVAL is nonzero, structural damping is used for direct or modal frequency or flutter analyses.

Entity: **WGHDRVA**

Entity Type: Relation

Description: Contains the user function requested element weight sensitivity information.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
MXCOL	Integer > 0	Matrix column number in [MXWGHDVA] for the sensitivity values

Created By: Module MSWGGRAD

Entity: **WGHRESP**

Entity Type: Relation

Description: Contains the user function requested element weight values.

Relation Attributes:

NAME	TYPE/KEY	DESCRIPTION
INSTANCE	Integer	User function instance number
POSITION	Integer Vector (2)	Response position index
VALUE	Real	Response value

Created By: Module MSWGRESP

Entity: **YS**

Entity Type: Subscripted Matrix

Description: Contains the column vector of enforced displacements of degrees of freedom constrained by single point constraints (see UG).

Matrix Form: A variable-sized single precision column vector having one row for each single point constraint degree of freedom.

Created By: Module MKUSET

Notes:

1. If no nonzero enforced displacements are specified for SPC'd degrees of freedom, this matrix will have no columns.

10. APPLICATION PROGRAMMER NOTES

One of the strengths of the ASTROS architecture is the ease with which the system can be modified and enhanced. Therefore, many ASTROS users will find it convenient and useful to write software to perform special purpose tasks within the ASTROS environment rather than in the more typical post-processing style. That is, instead of writing the ASTROS results to an intermediate medium to be read by the special routine, the new feature will be installed into the ASTROS system. The intent of this section is to give this type of ASTROS programmer an overview to writing modules that interface with the ASTROS system.

Coding standards and software design guidelines are given to create a module that functions within the ASTROS architecture without creating integration problems. Also, informal guidelines to using the ASTROS Dynamic Memory Manager (DMM) and the applications interface to the Computer Automated Design Database (CADDB) are presented. These presentations are necessarily limited in scope and do not provide a comprehensive guide to every aspect of ASTROS programming. They should, however, be sufficient for the advanced ASTROS user to write modules to perform special tasks without (1) duplicating functions that already exist in the ASTROS software and (2) writing code that causes errors outside the module being written. This section is divided into four subsections. Subsection 10.1 discusses coding standards including programming language and modular design. Subsection 10.2 contains suggestions for how to make the best use of the DMM, while subsection 10.3 is concerned with the CADDB features. Finally, Subsection 10.4 combines these topics and gives guidelines for I/O and system level interfaces within the ASTROS architecture.

10.1. SOFTWARE STANDARDS

ASTROS was designed to be machine transportable, maintainable, and easy to enhance. To achieve these goals, a set of software standards was defined both to design the modules and to guide the codification of the resulting algorithms. While these concepts were required for all ASTROS development efforts, they are not required for the developer of a new ASTROS module that is outside the scope of the official code. Many of these concepts, however, provide useful guidelines for all code development and avoid several potential pitfalls.

The most crucial concept is that of modularity. From the programmer's point of view, this means that the mechanisms by which the module communicates with other modules and with the executive are limited to maximize the independence of each module. For example, the interdependence of modules is reduced by communicating with data structures (e.g., relations, matrices, and unstructured entities) that reside on the database rather than in memory. Further independence is achieved by using dummy arguments to communicate between program units within a module and by avoiding the use of pathological communications such as common blocks. This mechanism of internal communication is well suited to the DMM for reasons of both modularity and simplicity. The DMM also eases the programmer's task in ensuring modularity by preventing the passage between modules of those data stored in open core blocks. The programmer is also given some additional debugging tools to ensure that the modularity has been maintained.

Modularity and maintainability may be further enhanced by recognizing that there are several distinct groups of code in ASTROS. These are the system level modules, the engineering application modules, the application utility modules, the large matrix utility modules, and the database/memory manager utility modules. All these modules comprising the current system are documented in Sections 3 through 8. Most important from the application programmer's point of view are the utility modules. The programmer should become familiar with the utility modules so that optimal use can be made of the existing ASTROS software. By using the existing utilities whenever possible, needless duplication of code is avoided and the maintainability is enhanced. Also, the ability to read the code is improved since the limited number of utilities are easily recognized by another programmer and no time need be spent in determining the function of a duplicate code block.

Machine transportability was a major consideration in the development of the ASTROS system. This requirement is not as critical for the application programmer who wants to install a special purpose module into the ASTROS system. It may be useful, however, to follow some basic guidelines, if for no other reason than the module's off-site utility. The basic guideline to follow is that all code be developed using the FORTRAN language adhering to the ANSI X3.9-1978 standard. This assures proper compilation and execution over a wide variety of computers and operating systems. Common nonstandard features such as certain file operations, FORTRAN enhancements and the use of site specific utility libraries should be avoided to maintain the transportability of the resultant system. It may be the case that the enhanced or site specific features are required in the new module. If so, the transportability of the system cannot be maintained: the programmer should give serious consideration to the ramifications of this loss before committing to a nonstandard piece of code.

In the author's experience, the most common problems with machine dependencies (aside from those already isolated) come from code developed using nonstandard FORTRAN enhancements. Often,

the resident FORTRAN compiler does not identify extensions to the standard and inexperienced programmers do not recognize that their code is nonstandard. A brief list of enhancements that are frequently seen and that should be avoided is:

1. The use of hollerith data outside DATA statements and FORMAT statements.

2. Special OPEN and CLOSE parameters and other file operations. Unless absolutely necessary, I/O should be limited to the database and the standard output file or punch file.

3. DO-WHILE and DO-UNTIL looping constructs — these can and should be emulated using standard FORTRAN.

4. Nonstandard variable names: in excess of 6 characters in length or containing nonstandard characters.

5. CHARACTER and non-CHARACTER data within a single named COMMON block — many compilers allow this without warning but it is nonstandard and can cause problems.

6. Very large or very small constants often create problems in that the range of legal real numbers is highly machine dependent.

7. Nonstandard type declarations: common examples are DOUBLE PRECISION COMPLEX, REAL*16, and INTEGER*1.

10.2. DYNAMIC MEMORY MANAGEMENT

ASTROS has a powerful and flexible Dynamic Memory Manager (DMM) which allows the programmer to allocate blocks of memory of a variable size determined at execution time. Various groups of memory, each containing one or more blocks, may be allocated and assigned names. The DMM then returns pointers to the various blocks relative to a single named common block unique to each module. All but the smallest local arrays used within a subroutine should use the DMM. There are three major reasons for this: (1) it eliminates the need to make *a priori* judgements about the size of arrays to allocate, thus eliminating or reducing the problem size restrictions (2) the DMM can automatically take into account the precision of the data and (3) the resultant code is more likely to be modular since the DMM promotes the modular use of memory resources. These important application utilities are documented in Section 8.3.

Memory can be allocated or released at any point in the code, allowing the programmer to make optimal use of system resources. Typically, however, memory requests are made in high level routines within a module, and the resultant memory block addresses are passed to the subroutines which make use of them. In this manner, the lower level routines may be coded in an open-ended, modular fashion with no need to keep track of memory block pointers. This approach is ideal when incorporating other codes (which do not make use of dynamic memory management) into the ASTROS system. In these cases, a top level driver routine for the new module may isolate all requests to allocate and subsequently free memory used within the module. By passing the starting address of the memory block to the lower level routines, the original array variables may often be preserved (thus avoiding extensive recoding) and any fixed dimensions for this array are removed. An equally effective technique is to pass the base core address (the address passed to the MMBASE utility) to the lower level routines. This method has the added benefit of eliminating the open core COMMON block from the lower level routines.

10.3. CADDB APPLICATION INTERFACE

CADDB is the basis of the majority of I/O activity within ASTROS. All of the input data, transitional results and answers are placed into entities on this database. Modules and subroutines are free to draw from the existing entities and to create new entities, both for archival purposes and to use as scratch space during execution. The CADDB applications interface has been designed to interact effectively with the DMM. For example, all entities include descriptions of the amount of data which they contain so that the proper memory block request can be made prior to retrieving the data. Techniques found in older systems, such as in NASTRAN, in which all available memory is allocated prior to entering the module, should not be used, although they may be emulated using the ASTROS DMM.

When incorporating other codes into the ASTROS system, it is desirable to conform to the ASTROS requirement that all input come from the input data stream and subsequently be available on the database. Although user input to a module can come from the MAPOL sequence or any external file, more typically, input will come from bulk data entries which are loaded at run time into corresponding database relational entities. The use of external files should be minimized because of the strongly machine dependent nature of the I/O interface and potential conflicts with the existing I/O interface. Hence, FORTRAN READ statements should not appear in any of the new modules. Instead, a suitable set of database entities should be designed such that existing READ statements can be replaced with queries to the database. As with the calls to the memory manager, described in Section 8.3, a common practice is to group the database calls in a high level or driver routine in a module and then pass the retrieved data to the lower level routines through parameter lists. This helps to keep the engineering routines free of the otherwise extraneous clutter which can result from a series of memory manager and database requests. This method is most effective when installing existing code as an ASTROS module.

Three types of database entities (relational, matrix, and unstructured) are used in ASTROS. Entities derived from the bulk data will typically be relational or matrix, while entities which are written in an engineering module can be of any type. Regardless of the type of entity, it must be opened with the DBOPEN database utility routine before any I/O operations are performed, and, when no more I/O will be done on the entity in the current module, it must be closed with the DBCLOS routine. Each entity type has its own suite of database management utilities, and these are documented in Section 8. The CADDB application programming interface is fairly complex and consists of a large number of individual utilities which may prove daunting to the novice ASTROS programmer — this is especially true of the matrix entity access methods. Section 8 is full of simple examples, however, which indicate the range of capability of the CADDB interface. The reader is strongly encouraged to study Section 8 and to write some simple routines to practice the interface. The experienced ASTROS programmer will find that the CADDB interface greatly simplifies the task of writing almost any piece of ASTROS software.

10.4. CODING PRACTICE — THE ASTROS INTERFACE

The architecture of the ASTROS system lends itself to enhancements. Nevertheless, the addition of new code is not a trivial task, and is further complicated by the flexibility of the MAPOL language to handle optional arguments, functions and subroutines, procedures and looping instructions, all of which play an important role in designing the module interface. The programmer is left with the difficult task of deciding how best to accomplish the interface in an environment so flexible that no clues are given at to how best to proceed. All that can be suggested is that existing modules be used as examples and that experience will shed light on the best methods to access modules.

Typically addition of new code is accomplished by adding a MAPOL callable module. This will be accompanied by the modification of several of the SYSGEN input files used for generating the ASTROS System database. These files are (1) the module definition file (MODDEF), (2) the relational schema file (RELATION), (3) the module control sequence (MAPOL), (4) the bulk data template file (TEMPLATE), and (5) the error message file (SERRMSG). The capitalized, parenthetical words represent typical root names of these files for ease of reference. The format for each of these files is specified in Section 3.2. The motivation behind each modification is described in the following notes:

MODDEF must be modified to define the new module to the system so that it may be called from the MAPOL sequence. If the module performs a function that may be generically applied to more than one database entity of the same type and form, it may then be desirable to include, with the module definition, calling argument definitions to pass in the names of the entities to be used.

RELATION must be updated to define the relational schemata of all the new relational database entities in the module, including those associated with new bulk data entries. Relational schemata can be defined instead through application calls to CADDB within the new module: however, this is cumbersome and may lead to problems when the module is executed more than once or not at all. Therefore, all relations should be defined through the SYSGEN RELATION file and all intermodular entities should be created by the executive (through a MAPOL declaration).

MAPOL must be changed if the new module is to be a part of the standard module control sequence. Otherwise, it will be necessary to include a new MAPOL sequence or an EDIT of the standard sequence in the input data file of every ASTROS job which will use the new module. In either case, the names and types of all new (and old) database entities which will be opened during a run must be declared in the MAPOL sequence which is used. However, a programmer should not modify the standard sequence unless he/she is very familiar with the ASTROS system. The MAPOL EDIT feature provides the necessary capabilities and avoids possible corruption of the standard sequence.

TEMPLATE must be revised to define new bulk data entries if the module requires any input data that cannot be supplied by the current bulk data entries and/or other engineering modules. Every template definition must have a corresponding entity definition to which the data are loaded.

SERRMSG must be updated to include any new error message texts that are to be accessed through the error message output utility, UTMWRT, documented in Section 6. Unless the module is to be made a permanent part of the ASTROS system, it may be preferable to write informa-

tional and error messages directly to the output file. As mentioned in the following, the unit number of the output file can be found in the /UNITS/ common block.

If communication is desired between subroutines within the new module, it is suggested that common blocks be avoided, especially common blocks containing fixed length arrays which place hard coded limits on problem size. Use of common blocks tends to obscure a module's structure and to reduce readability and maintainability. Instead, variables and arrays should be passed through the subroutines' calling lists. In cases where an older code has made prolific use of memory pointers (as in COSMIC/NASTRAN), there may be so many pointers to be passed that it is more convenient to place them within a common block than to pass them through argument lists. This approach can be justified by the fact that these pointers are set once in an upper level routine in the module and then never modified. If new common blocks are introduced, care must be taken to ensure that unique common block names are used.

For communication between modules, the database must be used. Common blocks must never be used for intermodular communication except for certain system level common blocks. The exception to the prohibition of common blocks is for common blocks which contain system parameters, such as FORTRAN I/O unit numbers, and mathematical constants, such as π. These quantities are so pervasive that requiring that they be passed through argument lists could be unduly burdensome. The following system level common blocks are used throughout ASTROS:

```
COMMON/CONDAS/      PI, TWOPI, RADEG, DEGRA
1                   FORPI2
COMMON/CONAD/       DPI, DTWOPI, DRADEG, DDEGRA,
1                   D4PISQ
COMMON/UNITS/       IUNIT (15)
COMMON/OUTPT1/      NLPP, MAXLIN, PAGENO, LINE,
1                   TOTLIN
COMMON/OUTPT2/      TITLE, SUBTIT, LABEL, HEAD1,
1                   HEAD2, HEAD3
```

/CONDAS/ and /CONDAD/ are set in block data routine ASTRBD and contain single and double precision versions, respectively, of trigonometric quantities. These are π , 2π , the multiplier from radians to degrees, the multiplier from degrees to radians, and $4\pi^2$. /UNITS/, /OUTPT1/, and /OUTPT2/ are set and documented in block data routine XXBD. /UNITS/ contains the FORTRAN file unit numbers for ASTROS I/O. IUNIT(1) is the ASTROS input file and IUNIT(2) is the ASTROS output file. /OUTPT1/ and /OUTPT2/ contain quantities used to control the formatting of data output. These two common blocks should be included in any utility module which uses the UTPAGE application utility module (documented in Section 6). NLPP represents the number of lines per page, MAXLIN is the maximum number of output lines allowed per ASTROS run, PAGENO is the current page number, LINE is the current line number, TOTLIN is the total number of lines output so far, and TITLE, SUBTIT, LABEL, HEAD1, HEAD2, and HEAD3 are the title, subtitle, lable, and headers which will be output in that order at the top of each page. Of these, LINE, HEAD1, HEAD2, and HEAD3 are the only variables that should be assigned by a routine which utilizes UTPAGE.