

Advanced CAE Applications for Professionals

Software that works — for you.SM

Installation Guide and System Support Manual for Version 20.1 Unix Operating Systems

 **UNIVERSAL ANALYTICS, INC.**

Publication GD-001

**© 1989-1999 UNIVERSAL ANALYTICS, INC.
Torrance, California USA**

All Rights Reserved

Restricted Rights Legend:

The use, duplication, or disclosure of the information contained in this document is subject to the restrictions set forth in your Software License Agreement with Universal Analytics, Inc. Use, duplication, or disclosure by the Government of the United States is subject to the restrictions set forth in Subdivision (b)(3)(ii) of the Rights in Technical Data and Computer Software clause, 48 CFR 252.227-7013.

The information contained herein is subject to change without notice. Universal Analytics Inc. does not warrant that this document is free of errors or defects and assumes no liability or responsibility to any person or company for direct or indirect damages resulting from the use of any information contained herein.

UNIVERSAL ANALYTICS, INC.

3625 Del Amo Blvd., Suite 370

Torrance, CA 90503

Tel: (310) 214-2922

FAX: (310) 214-3420

FOREWORD

This manual provides you with information on installing and configuring UAI Software Products. This suite of advanced software tools for Computer Aided Engineering includes:

- UAI/NASTRAN**
- ASTROS**
- eBase:applib**
- eBase:matlib**
- eShell**

The table below indicates which Chapters of this manual are important for the different UAI Software Products.

PRODUCT	CHAPTER								
	1	2	3	4	5	6	7	8	A
UAI/NASTRAN	✓	✓	✓	✓				✓	✓
ASTROS	✓	✓	✓		✓				✓
eBase:matlib	✓	✓	✓			✓			✓
eBase:matlib	✓	✓	✓			✓			✓
eShell	✓	✓	✓				✓		✓

This page is intentionally blank.

Table of Contents

FOREWORD	1-i
1. INSTALLATION GUIDE	1-1
1.1 AUDIENCE	1-1
1.2 DELIVERY MATERIALS	1-2
1.2.1 UAI/NASTRAN	1-2
1.2.2 ASTROS	1-2
1.2.3 eBase SOFTWARE PRODUCTS	1-3
1.3 COMPUTER SYSTEM REQUIREMENTS	1-4
1.4 DIRECTORIES ON THE DELIVERY MEDIA	1-5
1.5 INSTALLATION INSTRUCTIONS	1-6
1.6 INSTALLATION SUPPORT	1-11
1.7 SOFTWARE OR DOCUMENTATION PROBLEMS	1-11
1.8 ALTERNATE MACHINE INSTALLATION	1-11
2. CONFIGURING UAI PRODUCTS	2-1
2.1 AUDIENCE	2-1
2.2 WHAT IS A CONFIGURATION?	2-1
2.3 THE FORMAT OF PREFERENCE FILES	2-3
2.3.1 applib.pref Default Preference File	2-3
2.3.2 nastran.pref Default Preference File	2-3
2.3.3 eshell.pref Default Preference File	2-4
2.3.4 astros.pref Default Preference File	2-4
2.3.5 System, User, and Local Preference Files	2-4
2.3.6 Configuration Parameters	2-4

2.3.7	Nomenclature	2-5
2.4	THE CONFIGURATION SECTIONS	2-6
2.4.1	The Host Configuration Section	2-6
2.4.2	The eBase Kernel Configuration Section	2-6
2.4.3	The eBase:applib and eBase:matlib Sections	2-6
2.4.4	The eShell Configuration Section	2-6
2.4.5	The UAI/ NASTRAN Configuration Section	2-6
2.4.6	The ASTROS Configuration Section	2-6
2.5	EXAMPLE USE OF CONFIGURATION OVERRIDES	2-7
2.6	HOW APPLICATIONS MODIFY THE CONFIGURATION	2-7
3.	THE eBase DATABASE KERNEL	3-1
3.1	AUDIENCE	3-1
3.2	THE eBase PHYSICAL MODEL	3-1
3.2.1	Database Persistence	3-1
3.2.2	Database Physical and Logical Names	3-2
3.2.3	Where eBase Files Are Located	3-2
3.2.4	How an eBase is Named	3-2
3.2.5	Selecting Block Sizes for eBase Files	3-3
	Data Component Block Size	3-3
	Index Component Block Size	3-3
3.2.6	Distributing Database Files Across Multiple File Systems	3-3
3.3	OVERCOMING UNIX FILE SIZE LIMITS ON 32-BIT ARCHITECTURES	3-4
3.4	eBase DYNAMIC MEMORY	3-4
3.5	THE HOST and eBase CONFIGURATION SECTIONS	3-5
4.	THE UAI/NASTRAN PROGRAM	4-1
4.1	AUDIENCE	4-1
4.2	USING THE UAI/NASTRAN SOFTWARE COMPONENTS	4-1
4.2.1	The nastran Script	4-1
	The Log File	4-2
	Default Names for Output Files	4-2
	Modifying the nastran Script	4-3
4.2.2	The Demonstration Problem Library	4-3
4.2.3	The User's Guide Problem Library	4-3
4.2.4	The ALTER Library	4-3
4.2.5	The NEWS File	4-4
4.2.6	The SDRC Dataloader	4-4
4.2.7	The MSC/PATRAN Preference	4-5
4.2.8	The Online Manuals	4-5
4.3	DYNAMIC MEMORY MANAGEMENT	4-5
4.3.1	Sparse Matrix Solvers	4-6

4.4	UAI/NASTRAN PREFERENCE FILES	4-7
4.5	UAI/NASTRAN OVERRIDES OF eBase KERNEL CONFIGURATION	4-7
4.5.1	The UAI/NASTRAN Interface to eBase	4-7
4.5.2	Case Sensitivity of the ASSIGN Command	4-8
4.5.3	Overriding an eBase Location	4-8
4.5.4	Creating the eBase Name	4-10
4.5.5	Changing eBase Block Size Parameters	4-11
4.5.6	Assigning an Old eBase in UAI/NASTRAN	4-11
4.5.7	Automatic Assigning of Files	4-11
4.6	INTERFACE FILES FOR UAI/NASTRAN	4-13
4.7	THE UAI/NASTRAN CONFIGURATION SECTION	4-14
5.	THE ASTROS PROGRAM	5-1
5.1	AUDIENCE	5-1
5.2	USING THE ASTROS SOFTWARE COMPONENTS	5-1
5.2.1	The astros Script	5-1
	The Log File	5-2
	Default Names for Output Files	5-2
	Modifying the astros Script	5-3
5.2.2	The Applications Problem Library	5-3
5.2.3	The makelocalastros Script	5-3
5.2.4	The Online Manuals	5-4
5.3	DYNAMIC MEMORY MANAGEMENT	5-4
5.3.1	Sparse Matrix Solvers	5-5
5.4	ASTROS PREFERENCE FILES	5-6
5.5	ASTROS OVERRIDES OF eBase KERNEL CONFIGURATION	5-6
5.5.1	The ASTROS Interface to eBase	5-6
5.5.2	Case Sensitivity of the ASSIGN Command	5-7
5.5.3	Overriding an eBase Location	5-7
5.5.4	Creating the eBase Name	5-8
5.5.5	Changing eBase Block Size Parameters	5-9
5.5.6	Assigning an Old eBase in ASTROS	5-9
5.6	THE ASTROS CONFIGURATION SECTION	5-9
6.	APPLICATION PROGRAMMING INTERFACES: eBase:applib and matlib	6-1
6.1	AUDIENCE	6-1
6.2	USING THE eBase SOFTWARE SUITE	6-1
6.2.1	The Online Manuals	6-1
6.3	USING THE APPLICATIONS PROGRAMMING INTERFACES	6-2
6.3.1	Application Development	6-2
6.3.2	THE eBase:applib PROGRAMMING EXAMPLES	6-3
6.3.3	THE eBase:matlib PROGRAMMING EXAMPLES	6-4

6.3.4	Override Preference File Selection	6-4
6.4	API OVERRIDES OF eBase KERNEL CONFIGURATION	6-4
6.4.1	The applib Interface to eBase	6-4
6.4.2	Overriding an eBase Location	6-5
6.4.3	Creating the eBase Name	6-6
6.4.4	Changing eBase Block Size Parameters	6-7
6.4.5	Changing Other eBase Kernel Parameters	6-8
6.4.6	Using an Old eBase in the API	6-8
6.5	DYNAMIC MEMORY	6-8
6.5.1	Application Dynamic Memory	6-8
6.5.2	eBase:matlib	6-9
6.6	THE eBase:applib and eBase:matlib CONFIGURATION SECTIONS	6-9
7.	THE eShell PROGRAM	7-1
7.1	AUDIENCE	7-1
7.2	USING THE eShell PROGRAM	7-1
7.2.1	The eShell Program	7-1
7.2.2	The Tutorial Examples	7-2
7.2.3	The Online Manuals	7-2
7.3	eShell PREFERENCE FILES	7-3
7.4	eShell OVERRIDES OF KERNEL CONFIGURATION	7-3
7.4.1	The eShell Interface to eBase	7-3
7.4.2	Overriding an eBase Location	7-4
7.4.3	Creating the eBase Name	7-5
7.4.4	Changing eBase Block Size Parameters	7-6
7.4.5	Changing Other eBase Kernel Parameters	7-6
7.4.6	Using an Old eBase in eShell	7-6
7.5	eShell INTERFACE FILES	7-7
7.6	eShell DYNAMIC MEMORY	7-7
7.7	THE eShell CONFIGURATION SECTION	7-7
8.	UAI/NASTRAN GRAPHICS SOFTWARE	8-1
8.1	AUDIENCE	8-1
8.2	THE PLOTTING PROGRAMS	8-2
8.2.1	The Tektronix PLOT10 Plot Program	8-2
8.2.2	The PostScript Plot Program	8-2
8.2.3	The HP-GL Plot Program	8-3
8.2.4	The X Window System, Motif Interface Plot Program	8-4
8.2.5	Special Versions of the nastplot Program	8-6
8.2.6	Summary of Available Plotting Programs	8-7
8.3	THE UAI/NASTRAN PLOTTER COORDINATE SYSTEM	8-7
8.4	THE PLOTTER CHARACTER REPRESENTATION	8-7

8.5 THE PLOTTER COMMANDS	8-9
8.5.1 The <i>start_plot</i> Plot Operation	8-9
8.5.2 The <i>char</i> Plot Operation	8-10
8.5.3 The <i>draw_line</i> Plot Operation.	8-11
8.6 THE PLOT FILE FORMAT	8-11
8.6.1 The FORMATTED Plot File	8-11
8.6.2 The BINARY Plot File	8-12
APPENDIX A. THE CONFIGURATION SECTIONS	A-1
A.1 NOMENCLATURE	A-1
A.2 THE HOST CONFIGURATION SECTION	A-2
A.2.1 Site Description	A-2
A.2.2 Preference Override Information	A-2
A.3 THE eBase CONFIGURATION SECTION	A-3
A.3.1 Computing Resources	A-3
A.3.2 I/O System Parameters	A-5
A.3.3 Program Authorization	A-6
A.4 THE UAI/NASTRAN CONFIGURATION SECTION	A-6
A.4.1 Print File Controls	A-6
A.4.2 Computing Resources	A-7
A.4.3 Matrix Conditioning	A-8
A.4.4 Data Checking	A-9
A.4.5 Solution Techniques	A-11
A.4.6 Element Options	A-11
A.4.7 Analysis Output Control	A-12
A.4.8 I/O System Parameters	A-13
A.4.9 Assign Processing	A-13
A.4.10 Index Archive Control	A-14
A.4.11 Program Authorization	A-15
A.5 THE ASTROS CONFIGURATION SECTION	A-15
A.5.1 Print File Controls	A-15
A.5.2 Computing Resources	A-16
A.5.3 Matrix Conditioning	A-16
A.5.4 Data Checking	A-17
A.5.5 Solution Techniques	A-17
A.5.6 Element Options	A-18
A.5.7 I/O System Parameters	A-18
A.5.8 Optimization Control Options	A-18
A.5.9 Program Authorization	A-20
A.6 THE eBase:aplib CONFIGURATION SECTION	A-20
A.6.1 Computing Resources	A-20
A.7 THE eBase:matlib CONFIGURATION SECTION	A-21

A.7.1 Solver Options A-21

A.7.2 Timing Constants A-22

A.7.3 Program Authorization A-22

A.8 THE **eShell** CONFIGURATION SECTION A-23

A.8.1 Computing Resources A-23

A.8.2 Processing Defaults A-23

A.8.3 I/O System Parameters A-24

A.8.4 Program Authorization A-24

INDEX INDEX-1

1. INSTALLATION GUIDE

1.1 AUDIENCE

Although in many cases personnel responsible for the installation of UAI software products are quite knowledgeable about the software, there are other cases in which System Support specialists are responsible for the installation. This Chapter provides complete and detailed installation instructions which do not presuppose any UAI software expertise.

1.2 DELIVERY MATERIALS

This delivery includes one or more UAI software products that you have licensed. These products are described in the following sections.

1.2.1 UAI/NASTRAN

UAI/NASTRAN is proprietary software developed and maintained by Universal Analytics, Inc. **UAI/NASTRAN** software is available only to clients who have executed a Standard License or Evaluation Agreement with Universal Analytics, Inc. It is intended solely for the internal use of the Licensee.

The **UAI/NASTRAN** software is delivered either on tape media which depends on your specific computer, as described in Section 1.3, or over a network such as the Internet. Also delivered is the **UAI/NASTRAN** software documentation which consists of:

- The **UAI/NASTRAN** User's Reference Manual (Part No. ND-001)
- The **UAI/NASTRAN** User's Guide (Part No. ND-002)
- The **UAI/NASTRAN** Archive Schemata Manual (Part No. ND-003)
- This Installation and System Support Manual (GD-001)

Additional copies of these manuals may be purchased from Universal Analytics, Inc.

1.2.2 ASTROS

ASTROS is software originally developed by the USAF Wright Laboratory. It is marketed by Universal Analytics, Inc. under a Cooperative Research and Development Agreement (CRDA) with the government. In addition, certain enhancements to the program are proprietary to UAI. **ASTROS** software is available only to clients who have executed a Standard License or Evaluation Agreement with Universal Analytics, Inc. It is intended solely for the internal use of the Licensee.

The **ASTROS** software is delivered either on tape media which depends on your specific computer, as described in Section 1.3, or over a network such as the Internet. Also delivered is the **ASTROS** software documentation which consists of:

- The **ASTROS** User's Manual (Part No. AD-001)
- The **ASTROS** Programmer's Manual (Part No. AD-002)
- The **ASTROS** Theoretical Manual (Part No. AD-003)
- The **ASTROS** Applications Manual (Available from the USAF)
- This Installation and System Support Manual (GD-001)

Additional copies of these manuals, except for the last, may be purchased from Universal Analytics, Inc.

1.2.3 *eBase* SOFTWARE PRODUCTS

The Engineering Database Management System, *eBase*, includes three separate, but related, products: the *eShell* interactive shell, the *eBase:applib* Applications Programming Interface, and the *eBase:matlib* high-performance matrix utility library. Users licensing these products receive one or more of the following manuals:

- The *eShell* User's Manual (Part No. eB-001)
- eBase:applib* Programmer's Manual (Part No. eB-003)
- eBase:matlib* Programmer's Manual (Part No. eB-004)
- This Installation and System Support Manual (GD-001)

Additional copies of these manuals may also be purchased from Universal Analytics, Inc.

1.3 COMPUTER SYSTEM REQUIREMENTS

UAI software products require minimum hardware and software configuration for satisfactory installation and operation. This varies slightly from computer to computer. Requirements for delivery media and computer operating system level are shown in the following table:

COMPUTER	DELIVERY MEDIA	OPERATING SYSTEM ¹
Cray C90, J90, T90 Cray T90 IEEE	9-Track 6250 BPI Electronic	UNICOS 6.1.6
DEC Alpha	DDS Cartridge	DEC OSF 3.2
HP 9000/700 HP Exemplar	DDS Cartridge 8mm Cartridge	HP UX 9.05
IBM RS/6000	1/4" Cartridge or 8mm Cartridge DDS Cartridge	AIX 3.2
SGI (All)	1/4" Cartridge 8mm Cartridge DDS Cartridge	IRIX 5.3
SUN (All)	1/4" Cartridge 8mm Cartridge DDS Cartridge	SunOS 4.1

1. Minimum OS release level.

1.4 DIRECTORIES ON THE DELIVERY MEDIA

The directory structure of the delivery media is given in the following table.

DIRECTORY	DESCRIPTION
UAI/NASTRAN	
nastran20p1	Scripts, executable programs, and databases
nastran20p1/alterlib	DMAP ALTER library
nastran20p1/demolib	Demonstration Manual Input Data Streams
nastran20p1/patran	UAI/NASTRAN Preference file for MSC/PATRAN [®] Versions 5, 6 and 7.
nastran20p1/sdrc	SDRC Dataloader
nastran20p1/uguide	User's Guide sample input data streams
nastran20p1/utility	TEKPLOT Fortran source code
ASTROS	
astros20p1/bin	Scripts
astros20p1/exe	Executable programs and databases
astros20p1/lib	Object libraries
astros20p1/qaproblems	QA Problem data and results
astros20p1/samples	Sample test problem data
astros20p1/sysgendata	SYSGEN data
eShell	
eShell20p1	Scripts, executable programs, and databases
eShell20p1/tutorial	Tutorial problems
eBase:applib and eBase:matlib	
applib20p1	Libraries and support executables
applib20p1/applib	eBase:applib sample source files
applib20p1/matlib	eBase:matlib sample source files
INSTALLATION	
install20p1	Installation scripts and support executables
install20p1/check	Installation checking support data
ADOBE ACROBAT READER	
acrobat20p1	Installation file for Adobe Acrobat Reader used for viewing online documetation.
ONLINE DOCUMENTATION	
doc20p1	Online documentation files in Adobe Portable Document (PDF [®]) format.

1.5 INSTALLATION INSTRUCTIONS

The following steps must be performed to install UAI Software Products.





Note 1: To avoid compromising your system security, UAI product installation does not require **root** privileges. UAI strongly urges you not to install with those privileges.




Note 2: You must perform the installation on a machine which is licensed to run UAI products. If a tape drive is not available on the this machine, then Steps 1 and 2 may be done remotely to load the tape. After the tape is loaded, the remainder of the installation should be done on the licensed machine.

Step	Instructions								
1	<p>UAI recommends that install all UAI software products under a common directory tree. A good name for such a directory would be, for example, uaiapps. Within this directory there should be a subdirectory for each major release of the products. The current release is for version 20.x products, so the subdirectory should be called rel20. The installation procedures will automatically create subdirectories under rel20 for each product delivered. It will also handle any intermediate releases for the products such as 20.1, 20.2, and so forth. The commands to do this are:</p> <pre> mkdir uaiapps cd uaiapps mkdir rel20 cd rel20 </pre> <p>The amount of free space on the file system containing these directories depends on the products being installed. Conservative estimates for them are:</p> <table data-bbox="704 1411 1091 1512"> <tr> <td>UAI/NASTRAN</td> <td>40 Mbytes</td> </tr> <tr> <td>ASTROS</td> <td>60 Mbytes</td> </tr> <tr> <td>eShell</td> <td>10 Mbytes</td> </tr> <tr> <td>eBase:applib and matlib</td> <td>20 Mbytes</td> </tr> </table> <p>The name <i>rel_dir</i> will be used in the subsequent Chapters of this document to represent the rel20 directory recommended above. Also the names of the subdirectories created are simplified to be version independent. Thus, for example, the directory nastran20p1 is indicated by nastran#.</p>	UAI/NASTRAN	40 Mbytes	ASTROS	60 Mbytes	eShell	10 Mbytes	eBase:applib and matlib	20 Mbytes
UAI/NASTRAN	40 Mbytes								
ASTROS	60 Mbytes								
eShell	10 Mbytes								
eBase:applib and matlib	20 Mbytes								

Step	Instructions
2	<p>Before loading the files, make sure you move to the proper directory for this release of UAI Software Products, i.e. <code>cd rel20</code>. You must then load the UAI software delivery files using one of the following commands, depending on your computer system:</p>
	MOST SYSTEMS
	<pre>tar xvf /dev/tape_unit</pre>
	SILICON GRAPHICS ONLY
	<pre>tar xovf /dev/tape_unit</pre>
	IBM RS6000 WITH 8MM TAPE ONLY
	<p>You must make certain that the blocksize on your tape device is set either to 1024 or to 0. You may check the blocksize with the command:</p> <pre>lsattr -E -l tape_unit</pre> <p>To change the blocksize, use:</p> <pre>chdev -l tape_unit -a block_size=1024</pre> <p>Then, to load the tape, use:</p> <pre>tar xvf /dev/tape_unit</pre> <p>In all cases, <i>tape_unit</i> specifies the identifier of the tape device on which you have mounted the delivery tape. Several subdirectories will be created during the installation.</p>
	ELECTRONIC DELIVERY
	<p>In some cases, the delivery of UAI products may be performed electronically using <code>ftp</code>. Such a delivery consists of a single compressed <code>tar</code> file. After uncompressing the file using the <code>uncompress</code> utility, the commands shown above are used to load the individual UAI software delivery files. For example:</p> <pre>uncompress rel20p1.tar.Z tar xvf rel20p1.tar</pre>
	<p>The loading of the files in Step 2 will have created a subdirectory named install20p1 which contains the files necessary to complete your installation. All of the operations from Step 3 onward must be performed from that directory. Therefore, enter the command:</p> <pre>cd install20p1</pre>

Step	Instructions
	<p>In the next Step you will create the file named uaiuvf in the re120 directory. It is very important that you do not tamper with file uaiuvf in any manner. If the contents of this file are modified, if the file is moved from its current directory, or if its permissions are changed, UAI products will not function.</p> <p>If you inadvertently modify the uaiuvf file, you will have to repeat Step 3 of the installation.</p>
<p>3</p>	<p>The activation of UAI products can be accomplished using one of three procedures. If you are connected to the Internet, or if you have previously installed a UAI product, then a simple one-step procedure may be used. In other case, you must perform a two-step manual procedure requiring a contact with UAI. To execute the activation procedure, enter the command:</p> <pre>./activate</pre> <p>This interactive procedure will then give you a choice of the three activation methods. The following is required for each of these methods.</p> <p style="text-align: center;">Network Activation</p> <p>If successful, then no additional information is required, proceed to Step 4.</p> <p style="text-align: center;">Previously Activated</p> <p>You will be prompted for the name of the uaiuvf that was created in your previous installation. If successful, then no additional information is required, proceed to Step 4.</p> <p style="text-align: center;">Manual Activation</p> <p>The activate procedure creates a file in the re120 directory called uaiuvf. Then, it will provide you with an Activation Configuration Key, ACK.</p> <p>You must now call, FAX, or e-mail UAI and provide us with the ACK. A special FAX form, found on the last page of this Chapter, is available for you to use. The numbers are:</p> <p>U.S. Phone: 310-214-2922 U.S. Fax: 310-214-3420 e-mail activate@uai.com</p> <p>Ask for Installation Support when you call. You will be issued an Activation Verification Key, or AVK. The AVK is used in completing the Activation Procedure. You must now run the activation program a second time.</p> <pre>./activate</pre> <p>You will be prompted for your AVK. Note that the AVK is case insensitive so that you may enter any letters in either upper or lower case.</p>

Step	Instructions								
4	<p>The bulk of the remainder of the installation is performed with the setup procedure. This is executed with the command:</p> <pre data-bbox="852 359 963 384">./setup</pre> <p>This procedure performs the following tasks. Specific steps vary depending on the products that you are installing.</p> <ol data-bbox="802 499 1466 1066" style="list-style-type: none"> 1. The full path names of execution-related product files must be specified in some of the installed files. You will be prompted to enter path name information. All affected files will be automatically updated, and original copies of the changed files will be saved with the added file extension .orig. 2. The permissions of the installed files must be modified so that they are accessible to all of your users. You will be prompted for your choice of group or all other access permissions. 3. Several parameters must be updated to tell the products where temporary files and databases will be stored during execution of the programs. You will be prompted for these, and must be certain that you specify directories that have already been created. 4. A set of timing constants used by eBase:matlib routines is then computed for your computer. These are automatically stored in the appropriate preference file. 5. If the Adobe Acrobat Reader has not already been installed on your system, you will be given two choices. You can have the setup procedure install the reader for use only with the uaidoc command, or you can manually install it on your system so that it can be used any time. 								
5	<p>UAI Software products are delivered with a set of default Preference Files that control the function of the programs. These files are:</p> <table data-bbox="802 1171 1398 1270"> <tr> <td>UAI/NASTRAN</td> <td>nastran20p1/nastran.pref</td> </tr> <tr> <td>ASTROS</td> <td>astros20p1/astros.pref</td> </tr> <tr> <td>eShell</td> <td>eshell20p1/eshell.pref</td> </tr> <tr> <td>eBase:applib and matlib</td> <td>applib20p1/applib.pref</td> </tr> </table> <p>The parameters defined in these files are discussed in detail in the later Chapters of this manual. You should review these parameters to see if you wish to make specific changes for your site.</p>	UAI/NASTRAN	nastran20p1/nastran.pref	ASTROS	astros20p1/astros.pref	eShell	eshell20p1/eshell.pref	eBase:applib and matlib	applib20p1/applib.pref
UAI/NASTRAN	nastran20p1/nastran.pref								
ASTROS	astros20p1/astros.pref								
eShell	eshell20p1/eshell.pref								
eBase:applib and matlib	applib20p1/applib.pref								

Step	Instructions
	<p>The following Step may require certain system privileges depending on your site configuration.</p>
<p>6</p>	<p>To execute UAI products, the commands must be made available to your users. This can be accomplished in either of two ways.</p> <ol style="list-style-type: none"> 1. The commands can be linked to a directory in the path currently defined for users, this can be done with the command <pre>./linkbin</pre> <p>This procedure will prompt you for the directory where the links are to be placed and will then tell you exactly what it intends to do. Two links will be created for each command. The generic link, for example nastran, and the version specific link, nastran20p1. When newer versions of the software are installed, you will be prompted for permission to override the generic links, but the version specific links will be kept. Finally, version specific links for the new versions will always be created even if you choose to override the generic links. This allows you to access all versions of UAI products.</p> 2. The installation directories can be added to the user's path. This option requires more work because it must be done for each user who wishes to run UAI products. It does, however, have the advantage that root privilege is not required. <p>After you have made one of these choices, esh users must issue the rehash command or logout/login, to make the commands available. Bourne shell and korn shell users should be able to access the commands immediately.</p>
<p>OPTIONAL MSC/PATRAN[®] PREFERENCE INSTALLATION</p>	
<p>7</p>	<p>If you are installing UAI/NASTRAN and have purchased the MSC/PATRAN[®] Preference software, it may be installed if two conditions are met:</p> <ol style="list-style-type: none"> 1. The environment variable P3-HOME must be set and point to the directory in which your copy of PATRAN is installed. 2. The user ID used to run this script must be able to read from the UAI installation directories and write to the P3_HOME directory. <p>To run the installation procedure use the following command:</p> <pre>./installpatran</pre> <p>For a complete description of how to install the Preference and how to use it, see the file <code>/nastranp20/patran/install.txt</code>.</p> <p>The UAI PATRAN Preference requires a utility program, <code>uaiPAT3.out</code>, which reads the UAI/NASTRAN ARCHIVE database and writes the results to a PATRAN database. This program, as delivered, will only work correctly for PATRAN Version 6. If you are using the older Version 5, or the newest Version 7, of the software, then you will need to rebuild the utility. In both cases, instructions are also found in the file <code>/nastranp20/patran/install.txt</code>.</p>

1.6 INSTALLATION SUPPORT

If you encounter any problems during the installation procedure, call the UAI software product Hot-Line service at 310-214-2922 and ask for Installation Support.

1.7 SOFTWARE OR DOCUMENTATION PROBLEMS

If problems arise with any of the UAI products, please submit documentation of the problem to UAI to the attention of the Customer Support Department. Please submit the following data on a tape with your problem documentation.

- A Software Problem Report form, which is found in the front of the appropriate product manuals.
- Input data streams for **UAI/NASTRAN** or **ASTROS**.
- Archive and script files for **eShell** commands which caused problems.
- A copy of the Fortran code for the subroutine(s) that did not function correctly using **eBase:applib** or **eBase:matlib**.
- A full or partial copy of the database on which the error occurred. (remember that you may use the **EXPORT** function to make a partial copy). In the case of **UAI/NASTRAN** or **ASTROS**, any other files associated with the job should also be included.

Please use the same type tape as the one that originally contained your delivery materials. If this is not possible, please use one of the formats: 1/4" QIC-24, 1/4" QIC-150, 4mm DDS, or 8mm Exabyte. You may also use an IBM PC-compatible floppy disk if the data will fit. In all cases, please indicate the method used to create the tape or disk.

If it is necessary to include a smaller quantity of data, you may eMail the information to support@uai.com.

1.8 ALTERNATE MACHINE INSTALLATION

Most UAI Software Products contain a "node locking" security feature. In the event that you have the need to add new machines to your license agreement, or change machines under your current agreement, you will need to contact UAI to make arrangements for modification of the appropriate License Agreement. Also, you will need to provide UAI with certain identification information which will identify your computer(s) to the Software Product. The following table describes the procedures for obtaining this identification information as a function of computer manufacturer.

COMPUTER MANUFACTURER	COMPUTER MODEL	COMMAND	COMMENTS
Cray	C90, J90, T90 (Unicos)	<code>uname -a</code>	
DEC	Alpha	<code>/usr/sbin/uerf -r 300</code>	Report the ethernet hardware address of the form: xx:xx:xx:xx:xx:xx
HP	9000/7xx Exemplar	<code>uname -i</code>	
IBM	RS/6000	<code>uname -m</code>	
SGI	All Models	<code>sysinfo -s</code>	
SUN	All Models	<code>hostid</code>	

This page is intentionally blank.

2. CONFIGURING UAI PRODUCTS

2.1 AUDIENCE

This Chapter is intended for use by the person or group responsible for supporting the UAI software products within your organization. It provides you with general information on how UAI software products are configured.

2.2 WHAT IS A CONFIGURATION?

In general, UAI's suite of engineering software products uses computing resources intensively. As a result, there are many parameters that may be set to customize the software and to achieve optimal resource management on a given host computer. These parameters, taken as a group, are called the **Configuration** of the products. The configuration is provided through several **Preference Files**. These files include parameters which are used for controlling such things as database locations, physical file characteristics, memory utilization, and algorithm control.

For maximum flexibility, configurations may be controlled by the site, i.e. the UAI System Support Specialist, or the end user. Many different configurations may be defined for a site. For example, when configuring **UAI/NASTRAN**, the System Support Specialist may create different configurations for very small and for very large analyses.

The configuration of UAI products begins with the Default Preference Files included in your delivery. There are one or more of these files for each UAI product. One or more of the following files are used by each UAI Product:

`rel_dir/nastran#/nastran.pref` - **UAI/NASTRAN** parameters

`rel_dir/astros#/astros.pref` - **ASTROS** parameters

`rel_dir/eshell#/eshell.pref` – **eShell** parameters
`rel_dir/applib#/applib.pref` – **eBase** kernel, **applib** and **matlib** parameters

The parameters contained in each of these files may be overridden by the contents of other Preference Files that the user provides. The actual configuration used for a given execution is determined by:

- Processing the Default Preference Files and setting all parameters included in these files to their specified values and then applying the specified Preference Files in the following sequence:
- First, any selected System Preference File is processed, and any parameters included in it replace those previously defined. Typically System Preference Files are created by your System Support Specialist to define special parameters for different types of jobs run at your site. By default, System Preference Files are stored in the directory where UAI software products have been installed.
- Second, any selected User Preference File is processed, and any parameters included in it replace those previously defined. Typically users create User Preference Files to define special parameters that they wish to use for all of their jobs. By default, User Preference Files are stored in a user's home directory.
- Third, any selected Local Preference File is processed, and any parameters included in it replace those previously defined. Typically users create Local Preference Files to define special parameters that apply to only a single, or small number of, jobs. By default, Local Preference Files are stored in the current working directory from which the job is submitted.

In summary, the final configuration is the union of the Preference files. The Default Preference files contain a value for every parameter used by the product suite. The selection of System, User, and Local Preference Files is usually done with the command that invokes the program. See subsequent sections of this manual for how this is done for each product. The other Preference Files need only contain those parameters that differ from, and override, the default values. They can also override any of the parameters contained in the Default Preference Files.

Each Preference File is composed of several **Sections**. Each UAI software product may use one or more sections from each file. The sections used by UAI products are:

- The Host Section (**applib.pref**)
- The **eBase** Section (**applib.pref**)
- The **eBase:applib** Section (**applib.pref**)
- The **eBase:matlib** Section (**applib.pref**)
- The **UAI/NASTRAN** Section (**nastran.pref**)
- The **eShell** Section (**eshell.pref**)
- The **ASTROS** Section (**astros.pref**)

The format of the Preference File and a brief description of its various sections are described in the following sections.

2.3 THE FORMAT OF PREFERENCE FILES

A Preference File is a text file which is composed of several Sections as indicated above. Each Section includes a header followed by the parameters associated with the Section. For ease of use, each Section is divided into subsections which contain related parameters. The format of these files is shown in the following sections.

2.3.1 applib.pref Default Preference File

The `applib.pref` file consists of the sections shown below:

```
[Host]
  < Site Description >
  <Preference Override Information >

[eBase]
  < Computing Resources >
  < I/O System Parameters >
  < Program Authorization >

[eBase:applib]
  < Computing Resources >

[eBase:matlib]
  < Solver Options >
  < General MATLIB Options >
  < Timing Constants >
  < Program Authorization >
```

2.3.2 nastran.pref Default Preference File

The `nastran.pref` file consists is shown below:

```
[UAI/NASTRAN]
  < Print File Controls >
  < Computing Resources >
  < Matrix Conditioning >
  < Data Checking >
  < Analysis Output Control >
  < Solution Techniques >
  < Element Options >
  < I/O System Parameters >
  < Assign Processing >
  < Index Archive Control >
  < Entity Redirections >
  < Program Authorization >
```

2.3.3 `eshell.pref` Default Preference File

The `eshell.pref` file consists of the sections shown below:

```
[eShell]
  < Computing Resources >
  < Processing Defaults >
  < I/O System Parameters >
  < Program Authorization >
```

2.3.4 `astros.pref` Default Preference File

The `astros.pref` file consists of the sections shown below:

```
[ASTROS]
  < Print File Controls >
  < Computing Resources >
  < Matrix Conditioning >
  < Data Checking >
  < Solution Techniques >
  < Element Options >
  < I/O System Parameters >
  < Optimization Control Options >
  < Program Authorization >
```

2.3.5 System, User, and Local Preference Files

These files can contain any of the parameters defined in these Default Preference Files. They must, however, still contain the correct Section and Subsection definitions for each parameter defined:

```
[Section]
  < Subsection >
  ...
[Section]
  < Subsection >
  ...
...
```

2.3.6 Configuration Parameters

Configuration parameters are defined using one of the forms:

```
param_name = value
param_name = ( value,value,...,value )
```

The *param_names* are case-insensitive. The *values*, when character strings or floating point numbers with exponents, are also case-insensitive unless they are enclosed in single quotation marks as:

```
param_name = 'This is a Case-Sensitive String'
```

Only one parameter may be specified on each line of the file. Any characters that appear after *value* are treated as commentary and ignored. You may also enter comments into the file by beginning a line with any of the characters \$, *, or #.

2.3.7 Nomenclature

The Configuration Sections are described in detail in Appendix A. The table below, which for convenience is repeated in Appendix A, presents the symbols used in describing the Preference File parameters.

SYMBOL	DESCRIPTION
<i>char(len)</i>	Indicates that the parameter is a character string of maximum length <i>len</i> . Character strings need to be enclosed in single quotation marks only if they contain embedded blanks or special characters: MODEL=' IBM RS/6000 '
<i>int_val</i>	Indicates that the parameter is an integer value.
<i>real_val</i>	Indicates that the parameter is a real value. These values must be entered in decimal format, i.e. 10000000.0, or in exponential notation such as 1.0E+7.
<i>path_loc</i>	Indicates that the parameter is a Unix path name which must be enclosed in single quotation marks: Perm-eBASE-Data-Loc=' /big/disk/perm/storage '
<i>template</i>	Indicates that the parameter is a file name template. The asterisk in the template is the placeholder for the substitution.
<i>db_name</i>	Indicates that the parameter is a physical name for a database including any path information.
<i>file_name</i>	Indicates that the parameter is a physical name for a file.
<i>mem_val</i>	Indicates a memory value which can take one of the following forms xM words * 1000000 xMW words * 1000000 xMB bytes * 1000000 xK words * 1000 xKW words * 1000 xKB bytes * 1000 x words xW words xB bytes

Whenever a Unix path name is given, for either a directory or a file, two special forms may be used: The tilde (~) character can be used at the front of a path name to indicate the user's home directory. For example:

```
User-Pref-File-Loc = ~/*.pref
```

Also, Unix environment variables may be used in the path names. For example:

```
Temp-EBASE-Index-Loc = '$EBASETMP'  
Temp-EBASE-Data-Loc = '$EBASETMP'
```

2.4 THE CONFIGURATION SECTIONS

The following sections provide an overview of the sections in the Preference Files.

2.4.1 The Host Configuration Section

This section includes parameters that define the manufacturer and model of your host computer and licensing information. A complete description of available parameters is found in Section A.1.

2.4.2 The *eBase* Kernel Configuration Section

The *eBase* Configuration Section includes parameters which control the *eBase* Engineering Database Management System kernel. These include such information as default paths where databases are stored, physical block sizes for databases, and security information. This Section is discussed in detail in Chapter 3, and a complete description of all available parameters is found in Section A.2.

2.4.3 The *eBase:applib* and *eBase:matlib* Sections

The *eBase:applib* and *eBase:matlib* Configuration Sections are described in Chapter 6. The parameters in these sections include such items as dynamic memory sizes for *applib*, and timing constants for the *matlib* high-performance matrix utilities. A complete description of available parameters is found in Section A.2.

2.4.4 The *eShell* Configuration Section

The *eShell* Configuration Section includes parameters which control the *eShell* interactive interface to *eBase*. It includes such items as the *eShell* system database locations and dynamic memory specifications. Chapter 7 describes this section. A complete description of available parameters is found in Section A.5.

2.4.5 The UAI/NASTRAN Configuration Section

The UAI/NASTRAN Configuration Section includes parameters which control UAI/NASTRAN. These include controls on peripheral and computing resources, model data checking, program defaults, and so forth. This section is discussed in Chapter 4. A complete description of available parameters is found in Section A.3.

2.4.6 The *ASTROS* Configuration Section

The *ASTROS* Configuration Section includes parameters which control *ASTROS*. These include controls on peripheral and computing resources, model data checking, program defaults, and so forth. This section is discussed in Chapter 5. A complete description of available parameters is found in Section A.4.

2.5 EXAMPLE USE OF CONFIGURATION OVERRIDES

The following example shows a typical preference file that might be specified by **UAI/NASTRAN** users to define the resources necessary to run large models.

```
[eBase]
  < I/O System Parameters >
    Temp-eBase-Data-Loc = /work1/tmp
    Temp-eBase-Data-Loc = /work2/tmp
    Temp-eBase-Data-Loc = /work3/tmp

[UAI/NASTRAN]
  < Computing Resources >
    Working-Memory = 20mw
```

If the System Support Specialist stored these configuration parameters in a file named **bignast.pref** located in the installation directory, then users could select them with the following command:

```
nastran -ps bignast mydata
```

The configuration parameters found in **bignast** override those in the default Configuration file and provide more memory and disk resources for **UAI/NASTRAN**. The **-ps** parameter on the **nastran** command selects any override files. For complete descriptions of these parameters, see Chapters 3 and 4. The **nastran** command is described in detail in Chapter 6.

2.6 HOW APPLICATIONS MODIFY THE CONFIGURATION

All UAI Products and user-developed **eBase:applib** applications can override the default configuration parameters at run-time in three ways.

- ❑ By default, **eBase:applib** applications search for System, User, and Local override Preference Files having the name **uai.pref**. First the installation directory is searched, then the user's home directory, and finally, the current working directory. The names of these files, and their location, may be modified for your site by changing the Default Preference Files as described in Chapter 3.
- ❑ Users may provide their own file names and file locations. The exact method for doing this differs for each UAI Product. See the individual Chapters of this manual for specific instructions.
- ❑ The commands provided by each UAI product and the Applications Programming Interface used in the development of **eBase:applib** applications allows certain configuration parameters to be overridden. See the appropriate Chapters of this manual, and individual product manuals for additional details.

This page is intentionally blank.

3. THE *eBase* DATABASE KERNEL

3.1 AUDIENCE

This Chapter describes the basic concepts and configuration of the *eBase* kernel. Because these concepts are important to all *eBase* applications, including *UAI/NASTRAN*, *ASTROS* and *eShell*, support personnel should become familiar with this information.

3.2 THE *eBase* PHYSICAL MODEL

This section describes the physical *eBase* model in terms of the disk file structure and naming conventions. Logically, each *eBase* database may be viewed as a single entity. However, physically a database is composed of an *Index Component File* and one or more *Data Component Files*. The Index Component File has the fully qualified path:

```
index_loc/filename.edb
```

and each Data Component File has the fully qualified path:

```
data_loc/filename.ij
```

where *ij* are two hexadecimal digits which range from 00 to ff. The sections below explain the origin of the parts of the full path.

3.2.1 Database Persistence

Each database has a *Persistence*, which is either *Temporary* or *Permanent*. The user specifies the level of persistence when creating a database, and *eBase* uses the persistence to determine how it builds the full paths of the files which comprise the database.

3.2.2 Database Physical and Logical Names

Every database has a **Physical Name**, signified by *phys_name*, which is specified in different ways in each UAI product, as described in subsequent Chapters. The Physical Name is the basis for the *filenames* for each component file.

During the execution of each **eBase** application every database being used also has a **Logical Name** signified by *log_name*. This name is used by the application to identify the database during the execution of that program. A **Logical Name** is only connected to a **Physical Name** when it being used by the program. A different execution of the same or different program could then relate a different **Logical Name** to the same database. The methods used by each UAI product to connect a **Logical Name** to a **Physical Name** is different and are described in subsequent chapters.

3.2.3 Where eBase Files Are Located

Each UAI product has a method to specify the *index_loc*. If not specified, it defaults to the value of one of the Configuration parameters:

```
Perm-eBase-Index-Loc
Temp-eBase-Index-Loc
```

depending on the persistence of the database. As with the Index Component, each UAI product has a method to specify one or more *data_locs*. If not specified, these default to one of the sets of Configuration parameters:

```
Perm-eBase-Data-Loc
Temp-eBase-Data-Loc
```

also depending on the persistence of the database.

The number of *data_locs* provided by the **eBase** application determines the number of Data Component Files for that database. Each file is given a unique *ij* suffix. The above mentioned configuration parameters may be repeated once for each desired Data Component file to be used for the default case.

3.2.4 How an eBase is Named

For a Permanent database, the **eBase** kernel assumes that the user is careful about storing databases, and uses the Physical Name directly as the *filenames* in building the paths of the files.

For a Temporary database, the **eBase** kernel ensures that the *filenames* is unique among all others on that host computer system and as meaningful as possible. It consists of two parts concatenated together: a string no longer than six characters which identifies the database, and the five-digit Unix process identification number. If the Physical Name of the temporary database is six or fewer characters in length, it comprises the first part. Otherwise, **eBase** generates a unique string of six characters. The concatenated result, including the file suffix, never exceeds fourteen characters.



Because the uniqueness is based on both the database and process ID, the algorithm is designed to guarantee uniqueness only if you configure **eBase** on each host computer system with one or more temporary file locations which are not shared with the temporary file locations of **eBase** kernels on other host computer systems.

The user interface for defining database names is determined by the **eBase** application. Subsequent Chapters of this manual describe how this is done for **UAI/NASTRAN**, **ASTROS**, **eShell** and the **applib** and **matlib** Application Programming Interfaces.

3.2.5 Selecting Block Sizes for **eBase** Files

This section discusses the Block Sizes for the Index Component and the Data Components, how they are specified, and some considerations for their selection.

Data Component Block Size

The Data Components holds all the data stored in every entity on an **eBase** database. Changing the Data Component Block Size can have a dramatic impact on the database, both its size on disk and in memory, and the performance of I/O operations to store or retrieve information. Each entity which has been written requires at least one data block of disk storage. Each entity open for reading requires memory to hold one data block. Each entity open for writing requires memory to hold one or two data blocks simultaneously. Thus the trade-off should be clear: that increasing the block size improves the I/O performance for large entities, increases memory requirements for all open entities, and increases the disk space which small entities occupy.

Index Component Block Size

The Index Component holds all indexing information about the **eBase** database. This includes the directories, the entity subscripts, and the entity environment variables.

Choosing an optimal Index Component Block Size value means understanding the intended use of the database, and having a rough idea of the size. The minimum Index Component Block Size is 128 words.

3.2.6 Distributing Database Files Across Multiple File Systems

You may wish to distribute databases across multiple file systems. There are two reasons for doing this. The first is that databases may become very large, especially if they are created by an application using the **eBase:matlib**, such as **UAI/NASTRAN**. As a result, such applications may routinely fill all free space on a file system. By providing location data which points to several file systems, you allow these applications to spread data across the different file systems. The second reason is that overall throughput may be enhanced by allowing an application to perform I/O to a number of file systems. While executing, the application will create files on each of the file systems you specify, and then it will stripe the I/O

operations, i.e. it will write the first block to the first file system, the second to the second file system, and so on. This process continues in a round-robin fashion. Once a file system is full or the user's quota on it is exceeded, it is simply ignored and those not yet full are used.

You may also provide information to your users so that they may distribute databases as well by using the methods shown in subsequent Chapters of this manual.

If you plan to distribute your databases across several file systems, it is important that you locate the Index Component File on a different file system than the Data Component Files. If the file system that contains one of the Data Component Files becomes full during execution, the application will continue using the other file systems. An abnormal termination will not occur unless all file systems containing Data Component Files become full. However, since there is only one Index Component File, if the file system containing it becomes full, the application execution will fail immediately. When choosing the file systems that will contain the database files, you should remember that the Index Component File is quite small, generally less than one megabyte, whereas the Data Component Files can easily grow to hundreds of megabytes. This is especially true for large-scale computational systems such as **UAI/NASTRAN**.

3.3 OVERCOMING UNIX FILE SIZE LIMITS ON 32-BIT ARCHITECTURES

The **eBase** implementation under Unix on 32-bit architectures has a limitation due to the way *all* Unix programs access files. Files are byte-addressable, so the maximum size of a file is effectively limited to the number of bytes which can be represented in a signed integer. For 32-bit architectures, this is $2^{31}-1$, or approximately two gigabytes. No file may exceed this size. Traditionally, the Unix file systems themselves were also limited to this many bytes, but modern implementations have pushed the limit on the file system size, leaving the limit on individual files intact.

The ability to distribute an **eBase** database among multiple file systems discussed in the previous section also works to overcome the file size limit, because **eBase** places no restrictions on precisely where you locate the multiple Data Component Files. If you have a file system larger than a single file can ever become, you may overcome any limit on the size of an individual file by simply directing several Data Component Files to the same file system (or even the same directory). While the size of each Data Component File is limited under Unix, the size of the **eBase** Data Component as a whole is not.

3.4 **eBase** DYNAMIC MEMORY

The architecture of the **eBase** kernel allows for handling databases of virtually unlimited size. This is facilitated through the use of Dynamic Memory. This memory is typically much smaller than the working memory of an application. The main factor influencing the amount of database memory required is the block size used by the active databases. This is described in detail in subsequent sections of this chapter. The amount of memory used by **eBase** can be dynamically modified. Three configuration parameters:

eBase-Initial-Memory
eBase-Memory-Increment
eBase-Max-Memory

are used for this purpose. These are described later in this Chapter.

For these configuration parameters, you specify a complete *path_loc*.

3.5 THE HOST and *eBase* CONFIGURATION SECTIONS

The Host and *eBase* Configuration Sections, whose parameters define the host computer characteristics and default values the *eBase* kernel, are described in Sections A.2 and A.3.

This page is intentionally blank.

4. THE UAI/NASTRAN PROGRAM

4.1 AUDIENCE

This Chapter is intended for use by the person or group responsible for supporting the **UAI/NASTRAN** application in your organization. It provides you with information describing software resource utilization so that you may make changes to the **UAI/NASTRAN** procedures to tune the software for your computing environment and to set up the program in a manner that is familiar to your end-users. It also describes the additional libraries delivered with the program. This Chapter assumes familiarity with **UAI/NASTRAN** or other NASTRAN variants and an understanding of the *eBase* kernel information described in Chapter 3.

4.2 USING THE UAI/NASTRAN SOFTWARE COMPONENTS

4.2.1 The `nastran` Script

A *cs*h script file, called `nastran`, is provided to execute **UAI/NASTRAN**. To execute from the command line prompt, you enter:

```
nastran [ -m mem_val ] [-ps prefname]  
          [-pu prefname] [-pl prefname] filelist...
```

mem_val Specifies the working memory size to be used by **UAI/NASTRAN** for storing problem data. The actual units available to define the memory sizes are described in Chapter 2.

prefname Specifies the substitution string used to generate Preference File names. You may specify a different string for the System (`-ps`), the User (`-pu`) and the Local (`-pl`) preference files. If you have the unusual case where all of these files have the same name, you may use the option `-p` followed by the *prefname*.

filelist Specifies a list of one or more data file names, separated by spaces, that contain **UAI/NASTRAN** input data streams. By default, the script will search for a file with the name *filename.d*

The script file will execute **UAI/NASTRAN** using each of the data files that you provide. The print output is placed in a file which has the same name but having a trailing component of *.prt*. Other files may be automatically generated during an execution. These are described in the following two sections.

The Log File

During execution, **UAI/NASTRAN** writes information to a log file. This information is typically the history of modules as they execute within the program. The default log file name is:

filename.log

You, or your users, may monitor a job by viewing the log file periodically during execution. Also note that the Executive Control commands:

DIAG 8 and DIAG 19

may be used to print information during execution. By default, **UAI/NASTRAN** writes this information to the print file. However, if you or your users prefer to have these data placed in the log file, then you may modify the following configuration parameter:

Diag-Output=Log

At the end of the execution, the log file is normally appended to the print file and deleted. If desired, you may keep the log file as a separate file and not append it to the print file. The procedures for doing this are described below.

Default Names for Output Files

The **nastran** script procedure allows several files to be automatically created without having the user **ASSIGN** them explicitly. In these cases, the **UAI/NASTRAN** script will generate default filenames. These are:

Default for File with:	Is Assigned a Name of:	Whenever:
Standard PRINT	<i>filename.prt</i>	This file is always created.
USE=PUNCH or ASSIGN PUNCH=	<i>filename.pch</i>	The user has any Case Control output requests which direct solution results to the PUNCH device.
USE=BULK or ASSIGN BULK=	<i>filename.bulk</i>	The user has selected any Case Control commands which request the output of Bulk Data entries.

Modifying the `nastran` Script

The `nastran` script has provisions to allow the site to customize the trailing components of file names used by the script and the processing of the log file. The following environment variables are set at the top of the script which can be changed.

```
setenv DATAEXT d
setenv PRTEXT prt
setenv LOGEXT log
setenv APPENDLOG yes
```

The `DATAEXT` variable controls the trailing component of the files containing the input data streams. The default value is `.d` but you may wish to change this to `.dat`. Similarly the `PRTEXT` variable controls the trailing component of the output file, and `LOGEXT` controls the component of the log file. The processing of the log file is controlled by the `APPENDLOG` variable. If this variable has the value of `yes`, then the log file is appended to the end of the print file, and then it is deleted. If the `APPENDLOG` variable has the value `no`, then it is not appended to the end of the print file after the job finishes but is kept as a separate file.

4.2.2 The Demonstration Problem Library

The **UAI/NASTRAN** Demonstration Problem library may be found in the directory:

```
rel_dir/nastran#/demolib/
```

It consists of a set of complete input data streams. The names and descriptions of all these demonstration problems are provided in the **UAI/NASTRAN** Demonstration Problem Manual.

4.2.3 The User's Guide Problem Library

The **UAI/NASTRAN** User's Guide Problem library may be found in the directory:

```
rel_dir/nastran#/uguide/
```

This library consists of a set of complete input data files. The names and descriptions of all these demonstration problems are provided in the **UAI/NASTRAN** User's Guide.

4.2.4 The ALTER Library

UAI/NASTRAN includes an alter library which consists of packets of **DMAP** instructions. These alters relate to functions such as pre- and post-processor interfaces and prototype capabilities which have not yet been formally installed in the program.

Files from the alter library are requested by users with the **INCLUDE** Executive Control command:

```
INCLUDE ALTERLIB(alter_name)
```

where *alter_name* is the name of one of the files in the directory:

```
rel_dir/nastran#/alterlib/
```

You must define the directory where the **ALTERLIB** resides with the configuration parameter:

```
AlterLib-Loc
```

You are free to place any number of alter packets used by your organization into this directory. They may then be used from within **UAI/NASTRAN**.

4.2.5 The NEWS File

UAI/NASTRAN can be configured to print a **NEWS** section at the start of each print file. This data is stored in a simple text file. The printing of it is activated with the following configuration parameter:

```
News-File
```

As delivered, this file contains important user information relative to this version of **UAI/NASTRAN**. You may create your own News by storing the appropriate information in a new file and changing the configuration parameter to point to this file. The News output can be deactivated by deleting, or commenting out, the entry in the preference file.

4.2.6 The SDRC Dataloader

For clients of the SDRC I-DEAS software, UAI provides two customized versions of the Dataloader module, Versions 4 and 6. The version 6 Dataloader will also work with the SDRC Master Series Version 1.2 software. These versions have been optimized and tested for **UAI/NASTRAN** compatibility and several new capabilities have been added including the support of complex solution results. The executable files for these are found in the directory:

```
rel_dir/nastran#/sdrcl/
```

A script file, **nastdl**, has been provided to run the Dataloader. The script is self-documenting by running it with no parameters. The online help is:

```
Usage: nastdl [-v4/-v6] output2_file universal_file [option_codes]
        [first_case] [last_case]

output2_file  - name of UAI/NASTRAN OUTPUT2 file name
universal_file - name of Universal output file
option_codes  - AL,GD,AD,ND,ST,EN,NG,EC (Default=AL)
               multiple options are separated with commas
first_case    - first case to process (Default=1)
last_case     - last case to process (Default=999999999)
```

4.2.7 The MSC/PATRAN Preference

For users who use MSC/PATRAN[®], UAI provides an optional Preference. This Preference allows you to submit **UAI/NASTRAN** jobs directly from within PATRAN and to post-process the analysis results. Versions 5, 6 and 7 of PATRAN are supported. See Chapter 1 for instructions on installing the Preference.

4.2.8 The Online Manuals

The entire suite of **UAI/NASTRAN** manuals is available online in the Adobe Portable Document Format (PDF). This allows you to view the documentation on any computer that has the Adobe[®] Acrobat[®] Reader 3.0. If available from Adobe, the reader for your host computer, or computers, was delivered with your system. Any other readers that become available can be downloaded from the Adobe Web site at www.adobe.com.

To use the documents, from the command line you enter:

```
uaidoc [manual_name]
```

If you omit the *manual_name*, then you will see a splash screen that allows you to navigate to the appropriate manual. You may also go directly to a manual by placing its name on the command lines. The names of the **UAI/NASTRAN** manuals are:

- nastran_ref*
- nastran_guide*
- nastran_schema*
- system_support_unix*

Check the UAI Web site at www.uai.com for any interim updates and additions to the electronic documentation.

4.3 DYNAMIC MEMORY MANAGEMENT

The architecture of **UAI/NASTRAN** allows the analysis of finite element models of virtually unlimited size. Most numerical calculations perform at maximum efficiency when all data for the operation fits in the working memory space of the program. Many operations may be performed even when all data that they require do not fit in memory by using what is called **spill logic**. Spill logic simply involves the paging of data to and from disk storage devices as necessary. For very large jobs, spill commonly occurs. In such cases, providing **UAI/NASTRAN** with additional memory can often improve performance. This is especial true when using the high-performance sparse matrix solvers. For these algorithms, maximum memory is important for optimal performance. This is discussed in the next section.

Some performance problems can develop on virtual memory machines where the working memory used by **UAI/NASTRAN** exceeds the physical memory available in the machine. This can happen quite often with the high-performance sparse solvers that have large working memory require-

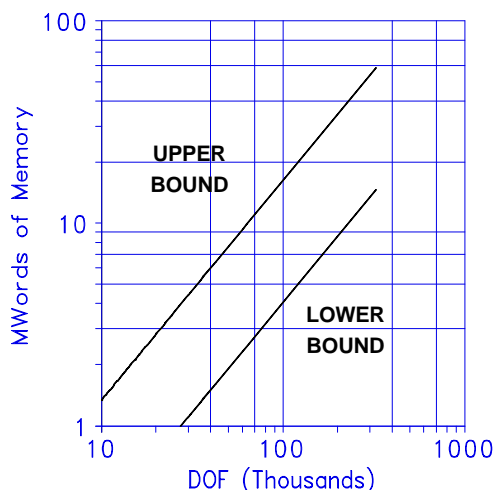
ments. Certain matrix algorithms, in particular some of the **MPYAD** and **FBS** methods, will cause excessive system paging which will degrade system performance. In these cases the user can use the **PHYSICAL** option on the **MEMORY** Executive Control Command or the configuration parameter `Physical-Memory` to restrict these algorithms to only use a portion of the working memory. This memory restriction will only be used for those algorithms that have shown to cause excessive paging. Other methods, such as the high-performance sparse solvers that function well in a virtual memory environment, will not be limited. The exact value to use on these options must be determined by the user because it is affected by the amount of physical memory available to your jobs and the current load on the system.

The working memory for **UAI/NASTRAN** is dynamically acquired during execution. The amount of space that is actually used by the program is determined, in order of precedence, by the **MEMORY** Executive Control Command, the `-m` option of the `nastran` script, and the configuration parameter `Working-Memory`.

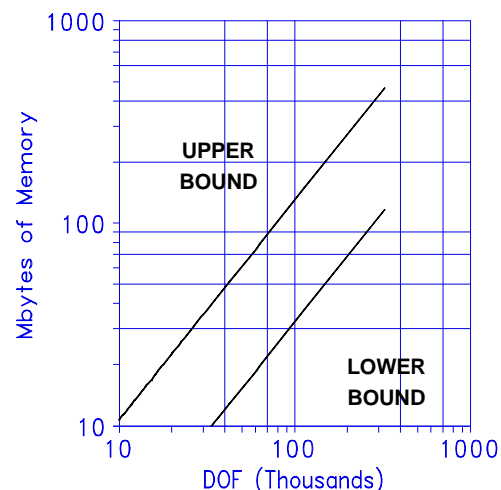
Some Unix machines require special actions when it is necessary to execute **UAI/NASTRAN** with very large working memory values. These actions may require rebuilding the Unix kernel, or changing the UAI software executables themselves. Please contact UAI if you experience problems running very large jobs.

4.3.1 Sparse Matrix Solvers

UAI/NASTRAN has two high-performance solvers which take advantage of the latest developments in sparse matrix algorithm technology. The first of these is the symmetric matrix decomposition used in static analyses, and the second is the Lanczos eigen extraction method. This latter method is used for extracting a modest number of eigenvalues from very large systems. When these solvers are used, memory requirements may become significant. The figures on the next page give upper and lower bound estimates for the amount of memory that users should specify on their **MEMORY** Executive Control Command. Note that if users do not specify enough memory for the new algorithms, the program will revert to the standard solution algorithms. Also note that these curves have been created using a representative sample of real analysis jobs. They are intended only to be used as guidelines — a specific job may take significantly more or less memory than indicated. On virtual memory machines, you should not be concerned if the required working memory exceeds the actual physical memory installed on the machine. These solvers have been optimized to run efficiently in this environment.



Cray



Others

4.4 UAI/NASTRAN PREFERENCE FILES

As discussed in Chapter 2, *eBase* provides System, User and Local Preference Files that can be used to override the parameters in the Default Preference File. In UAI/NASTRAN, the `-ps`, `-pu`, and `-pl` options of the `nastran` script are used to request this feature. These values are the substitution strings used by *eBase* to generate the actual file names. If these options are not used, then *eBase* will search for System, User, and Local override Preference Files having the name `uai.pref`. First the installation directory is searched, then the user's home directory, and finally, the current working directory. The names of these files, and their location, may be modified for your site by changing the Default Preference Files parameters in the `Host` section as described in Chapter 3.

4.5 UAI/NASTRAN OVERRIDES OF *eBase* KERNEL CONFIGURATION

The UAI/NASTRAN I/O subsystem is based on the *eBase* Kernel described in Chapter 3. There are three kinds of UAI/NASTRAN databases. The first is the database used to store transient data during execution. This is called the *Run-Time Database*, `RUNDB`. The second kind of database is the Permanent database which may be used to save data between UAI/NASTRAN executions. There are three *Permanent Databases*: the `SOB` Database, which is used when performing substructuring analyses; the `NLDB` Database which is used when performing material nonlinear and GAP element analyses; and the `ARCHIVE` database which contains UAI/NASTRAN model and results data in relational form. The third kind of database is the *System Database* used by UAI/NASTRAN to store internal data. This database is delivered with the system and is only used in a Read-Only access mode by the program.

4.5.1 The UAI/NASTRAN Interface to *eBase*

The UAI/NASTRAN user interface to databases is simple. As shown in the UAI/NASTRAN User's Manual, the `ASSIGN` command is used to define databases:

```

ASSIGN db_name [ , { NEW } ] , USE = { SOF
              [ , { OLD } ]           [ NLDB
              [ , { TEMP } ]          [ RUNDB
                                      [ ARCHIVE ]
[ , REALLOC ] [ , PASSWORD = pass ]

```

Actually, there are additional parameters that are available to users when assigning database files. The complete general form of the command:

```

ASSIGN db_name [ , { NEW } ] , USE = { SOF
              [ , { OLD } ]           [ NLDB
              [ , { TEMP } ]          [ RUNDB
                                      [ ARCHIVE ]
[ , REALLOC ] [ , PASSWORD = pass ]
[ , IBLKSIZE = int_val ] [ , ILOC = index_dir ]
[ , DBLKSIZE = int_val ] [ , DLOC = db_dir_list ]

```

Each of these command parameters is used to override the **eBase** kernel configuration parameters as described in the following sections.

The *db_name* parameter has one of the following formats:

```

logical_name
logical_name = phys_name
logical_name = phys_loc/phys_name
logical_name = phys_loc/

```

4.5.2 Case Sensitivity of the ASSIGN Command

Previous versions of **UAI/NASTRAN** automatically converted all input data to upper case including any Executive Control **ASSIGN** and **INCLUDE** commands. This required the use of single quotation marks around the file name if it was desired to be in lower case. Starting in version 11.7 the site can control this behavior with the `Upper-Case-Assign` parameter. If the value of this parameter is `No` then the file name portion of any **ASSIGN** and **INCLUDE** commands will not be converted to upper case. All other portions of the command are case insensitive with the exception of the logical name. For the purposes of matching logical names in the data stream, logical names are also case insensitive. But if the logical name is used to generate the physical file name, because it is omitted, then the actual case entered is used.

4.5.3 Overriding an eBase Location

The directory where database files are stored comes from one of three places: the directory information in the *phys_loc* field of the *db_name* parameter; the **ILOC** and **DLOC** parameters; or the **eBase** Configuration Section. These methods apply to both Temporary and Permanent databases.

The directory information in the *phys_loc* field provides a convenient method to place a database when it is desired to have both the Index and Data Component in the same directory and only one Data Component file is required. It is illegal to use this method if you have selected either an **ILOC** or **DLOC** parameter. The parameter:

```
ILOC=path_loc
```

specifies the full path name of a directory where you want to store the Index Component File of the database being **ASSIGNED**. This applies whether the database exists or is being created. Similarly, either form of the parameter:

```
DLOC=path_loc
DLOC=(path_loc1,path_loc2,...)
```

specifies the full path name of one or more directories where you want to store the Data Component Files of the database being **ASSIGNED**.

If no directory information is provided with the *phys_loc* field, and neither **ILOC** nor **DLOC** are specified, then the **eBase** Configuration parameters discussed in Chapter 3 are used.

For example, to **ASSIGN** a temporary **RUNDB** Database whose Index and Data Components will be placed in the directory */usr1/tmp*:

```
ASSIGN RUNDB=/usr1/tmp/,TEMP,USE=RUNDB
```

When your users are using **SOF** or **NLDB** databases, they may also distribute the data across multiple file systems. Consider the example to **ASSIGN** a Permanent **SOF** Database whose Index Component will reside in */usr1/tmp* and whose Data Components will reside in */usr2/tmp*, */usr3/tmp*, and */usr4/tmp*:

```
ASSIGN SOF=mysof,NEW,USE=SOF,PASSWORD=pass,
        ILOC=/usr1/tmp,
        DLOC=(/usr2/tmp,/usr3/tmp,/usr4/tmp)
```

4.5.4 Creating the *eBase* Name

The **UAI/NASTRAN** databases follow the general naming conventions described in Chapter 3. The naming conventions for the physical files differ for Permanent and Temporary databases. This is illustrated in the following table.

eBase FILE NAMES GENERATED BY VARIOUS FORMS OF <i>db_name</i>	
<i>logical_name</i> =/ <i>phys_loc</i> / <i>phys_name</i> or <i>logical_name</i> = <i>phys_name</i>	<i>logical_name</i> or <i>logical_name</i> =/ <i>phys_loc</i> /
FOR PERMANENT DATABASES	
<i>phys_name</i> .*	<i>logical_name</i> .*
FOR TEMPORARY DATABASES	
<i>phys_name</i> xxxxx.*	<i>logical_name</i> xxxxx.*
if <i>phys_name</i> or <i>logical_name</i> is longer than six characters <i>unique</i> xxxxx.*	
The notation ".*" indicates that all of the database files, both index and data, have the specified name.	

As shown, to specify a name, the *db_name* parameter of the **ASSIGN** command is used. In its simplest form, this parameter gives just the Logical Name of the *eBase* database and this name is also used to generate the physical file names. If the *db_name* parameter also contains a *phys_name* field then this name is used instead. The general form of the *db_name* parameter was discussed in a previous section of this chapter.

The name generation for temporary databases follows the same general rules as for permanent databases except the *eBase* kernel will also append the five character Unix process id on the end of the name. This is to insure that the temporary file names will be unique for all jobs running simultaneous on the machine. Also to insure that the temporary file names are valid on all versions of Unix the total file name will not exceed fourteen characters. Therefore the *logical_name* or *phys_name* field used should not contain more than six characters. If it does, then *eBase* will automatically generate a unique six character name to use instead.

You can also use the asterisk (*) character in the *phys_name* portion of the *db_name* parameter. In this case, the asterisk is replaced by the name of the input data file with any file extensions removed. For example:

```
ASSIGN SOF=* .SOF,NEW
```


4.5.5 Changing *eBase* Block Size Parameters

Database block sizes are also controlled by the *eBase* Configuration Section or the `DBLKSIZ` and `IBLKSIZ` parameters. The two `ASSIGN` command parameters:

```
IBLKSIZ=int_val
DBLKSIZ=int_val
```

are used to control the size of physical blocks used for performing I/O operations. The first, `IBLKSIZ`, specifies the block size for the Index Component, and the second parameter, `DBLKSIZ`, specifies the block size of the Data Component. All block size values are specified in single precision words. For example, to `ASSIGN` a temporary `RUNDB` whose Data Component block size is 8192 words:

```
ASSIGN MYRUNDB,TEMP,USE=RUNDB,DBLKSIZ=8192
```

While there are no restrictions on the block sizes of the various databases `ASSIGNED` during a `UAI/NASTRAN` execution, when using Automated Substructuring or Nonlinear analyses, it is more efficient if the `SOF` and `NLDB` databases have the same Data Component block sizes as the `RUNDB`.

4.5.6 Assigning an Old *eBase* in `UAI/NASTRAN`

To `ASSIGN` a database that was created in another `UAI/NASTRAN` execution, the disposition of `OLD` is used. It is not necessary to respecify all of the characteristics that were used when the database was created, because this information is saved on the database. These characteristics include blocks sizes and directory paths for the data components of the database. The only parameters that must be specified are the physical name, the directory path to the index file, and the correct password. This is illustrated in the following example:

When the database is created:

```
ASSIGN SOF,NEW,PASSWORD=pass,
      DLOC=(/usr1/tmp./usr2/tmp),
      IBLKSIZ=256,DBLKSIZ=8192
```

When the database is reused:

```
ASSIGN SSOF,OLD,PASSWORD=pass
```

4.5.7 Automatic Assigning of Files

`UAI/NASTRAN` provides a facility to allow the site to predefine a set of `ASSIGN` commands that will be automatically used when required. These are defined using the *Auto-Assign* entries in the preference file. Up to fifty of these may be defined. The following examples show what these entries may look like.

```
Auto-Assign='output2=*.out2,new,realloc,
             use=output2,type=binary'
Auto-Assign='plot=*.plt,new,realloc,use=plot'
```

The values of these parameters are templates that are expanded to form a valid **ASSIGN** command. The expansion involves replacing the asterisk with the name of the file containing the input deck minus any extensions. The case of the template is only important for the file name portion. All other parameters are case insensitive.

The appropriate entry is chosen by searching the list and selecting the first matching entry. For most types of **ASSIGNs** this will be the entry with the matching logical unit name and correct **TYPE** parameter. The following commands, which request **OUTPUT2** formatted output, would select the first of the above Auto-Assign requests.

```
OUTPUT2 UGV//-1/*OUTPUT2* $
POST SDRC TO OUTPUT2
```

For units like **SOF**, **NLDB**, **RUNDB**, **PLOT**, **CHKPNT** and **DICT** which are not requested by logical name, the first entry with the correct **USE** value is selected. In the above examples the second Auto-Assign request is used for any structural or X-Y plot request.

To make the automatic **ASSIGN** feature more usable for **INPUTT_x** and **OUTPUT_x** units, an additional set of configuration parameters is available to set the default value for the logical unit names to be used by these modules. This is desired because previous versions of **UAI/NASTRAN** used the same default logical name of **USER1** for all of these modules. The following parameters are available.

```
Logical-Unit-Inputt1 = logical_name
Logical-Unit-Output1 = logical_name
Logical-Unit-Inputt2 = logical_name
Logical-Unit-Output2 = logical_name
Logical-Unit-Inputt4 = logical_name
Logical-Unit-Output4 = logical_name
Logical-Unit-Inputt5 = logical_name
Logical-Unit-Output5 = logical_name
```

4.6 INTERFACE FILES FOR UAI/NASTRAN

In addition to databases, **UAI/NASTRAN** may also create a number of interface files which are used to communicate with other programs or to save data from one execution to another.

The general form of the **ASSIGN** command for these files is:

$$\text{ASSIGN } \textit{logical_name} [= \textit{phys_name}] \left[\left\{ \begin{array}{l} \text{NEW} \\ \text{OLD} \\ \text{TEMP} \end{array} \right\} \right] [, \text{USE} = \textit{use}]$$

$$[, \text{REALLOC}] \left[\left\{ \begin{array}{l} \text{BINARY} \\ \text{FORMATTED} \end{array} \right\} \right] [, \text{BLKSIZE} = \textit{int_val}]$$

You can also use the asterisk (*) character in the *phys_name* portion of the *db_name* parameter. In this case, the asterisk is replaced by the name of the input data file with any file extensions removed.

The table below summarizes the allowable **TYPE**s for each **USE**.

IF USE IS:	THEN TYPE MAY BE:	
	BINARY	FORMATTED
CHKPNT	✓	
DICT		✓
IMPORT	✓	
INPUTT1	✓	
INPUTT2	✓	
INPUTT4	✓	✓
INPUTT5	✓	✓
OUTPUT1	✓	
OUTPUT2	✓	
OUTPUT4	✓	✓
OUTPUT5	✓	✓
PLOT	✓	✓
RESTART	✓	
SOFIN	✓	✓
SOFOUT	✓	✓

The command parameter:

BLKSIZE=*int_val*

is used to control the size of physical blocks used for performing I/O operations. You may control your system default for this value with the configuration parameter:

```
External-File-BlockSize = int_val
```

The `External-File-BlockSize` default value should always be adequate for all but the largest jobs. All block size values are specified in single precision words.



The directory where external files, **ASSIGNED** with a disposition of **NEW** or **OLD**, are stored is obtained from the directory information in the `phys_name` parameter. If no directory information is specified, the location is the user's current directory.

External files **ASSIGNED** with a disposition of **TEMP**, are placed in the directory specified by the Configuration parameter:

```
External-Temp-Dir = 'your_path'
```

To **ASSIGN** an external file that was created in another **UAI/NASTRAN** execution, the disposition of **OLD** is used. All of the characteristics used in the original run must be specified in exactly the same manner when you **ASSIGN** the file in any subsequent run. This includes the Physical File name, type, block size parameters, and all directory paths. Additionally, if a binary **RESTART**, **INPUTT1** or **SOFIN** file is used, the current **RUNDB** Data Component block size must be greater than or equal to the original **RUNDB** Data Component block size from the run that created the files.

4.7 THE UAI/NASTRAN CONFIGURATION SECTION

The **UAI/NASTRAN** Configuration Section, whose parameters define default values for system and engineering data which are unique to **UAI/NASTRAN**, are described in Section A.4.

5. THE *ASTROS* PROGRAM

5.1 AUDIENCE

This Chapter is intended for use by the person or group responsible for supporting the *ASTROS* application in your organization. It provides you with information describing software resource utilization so that you may make changes to the *ASTROS* procedures to tune the software for your computing environment and to set up the program in a manner that is familiar to your end-users. It also describes the additional libraries delivered with the program. This Chapter assumes familiarity with *ASTROS* and an understanding of the *eBase* kernel information described in Chapter 3.

5.2 USING THE *ASTROS* SOFTWARE COMPONENTS

5.2.1 The *astros* Script

A *sh* script file, called *astros*, is provided to execute *ASTROS*. To execute you enter:

```
astros [ -m mem_val ] [-ps prefname]  
          [-pu prefname] [-pl prefname] filelist...
```

mem_val Specifies the working memory size to be used by *ASTROS* for storing problem data. The actual units available to define the memory sizes are described in Chapter 2.

prefname Specifies the substitution string used to generate Preference File names. You may specify a different string for the System (-ps), the User (-pu) and the Local (-pl) preference files. If you have the unusual case where all of these files have the same name, you may use the option -p followed by the *prefname*.

filelist Specifies a list of one or more data file names, separated by spaces, that contain **ASTROS** input data streams. By default, the script will search for a file with the name *filename.d*

The script file will execute **ASTROS** using each of the data files that you provide. The print output is placed in a file which has the same name but having a default trailing component of *.prt*. Other files may be automatically generated during an execution. These are described in the following two sections.

The Log File

During execution, **ASTROS** writes information to a log file. This information is typically the history of modules as they execute within the program. The default log file name is:

filename.log

You, or your users, may monitor a job by viewing the log file periodically during execution.

At the end of the execution, the log file is normally appended to the print file and deleted. At that time, the log file is filtered to exclude individual module histories and give only the summary information. If **ASTROS** terminates abnormally, however, the unfiltered log file is retained in the local directory.

Default Names for Output Files

The **astros** script procedure allows several files to be automatically created without having the user **ASSIGN** them explicitly. In these cases, the **ASTROS** script will generate default filenames. These are:

Default for File with:	Is Assigned a Name of:	Whenever:
Standard PRINT	<i>filename.prt</i>	This file is always created.
USE=PUNCH or ASSIGN PUNCH=	<i>filename.pch</i>	The user has any Solution Control output requests which direct solution results to the PUNCH device.
Message Queue	<i>filename.err</i>	Queue file for messages.

Modifying the `astros` Script

The `astros` script has provisions to allow the site to customize the trailing components used by the script. The following environment variables are set via the `astrosextensions` script in the directory:

```
rel_dir/astros#/bin
```

The `astrosextensions` script may be modified to change the defaults.

```
ASTROS_ERREXT="err";export ASTROS_ERREXT
ASTROS_INPEXT="d";export ASTROS_INPEXT
ASTROS_PCHEXT="pch";export ASTROS_PCHEXT
ASTROS_PRTEXT="prt";export ASTROS_PRTEXT
ASTROS_LOGEXT="log";export ASTROS_LOGEXT
```

The `ERREXT` variable controls the trailing component of the message queue file. The `INPEXT` variable controls the trailing component of the files containing the input data streams. Similarly the `PRTEXT` and `PCHEXT` variables control the trailing component of the output file and the punch file, respectively. Finally, the `LOGEXT` variable controls the extension of the log file. The default values are shown in the list.

5.2.2 The Applications Problem Library

The *ASTROS* Applications Problem library may be found in the directory:

```
rel_dir/astros#/applications/
```

It consists of a set of complete input data streams. The names and descriptions of all these demonstration problems are provided in the *ASTROS* Applications Manual available from the United States Air Force.

5.2.3 The `makelocalastros` Script

You may extend *ASTROS* program by writing and embedding software into the executable program. This process is fully documented in the *ASTROS* Programmer's Manual. When new versions of the program are needed, you will use the `makelocalastros` script to compile and link the Fortran or C language components with the delivered libraries to create a new, local, version of *ASTROS*.

To use the script, you must have all the local software components (`.f` and/or `.c` files) that must be linked located in one directory. From within that directory you issue the command:

```
makelocalastros >& make.lis& (if csh) or
makelocalastros >make.lis 2>&1& (if sh or ksh)
```

If all your code compiles successfully, the result will be a new version of the file `astros.out` that can be used with the `astros` script to be the `astros` executable.

By default, every `.f` and/or `.c` file will be compiled using the `-debug` options in the delivered scripts `astrosf77` or `astrosc`. To avoid the `-debug` options, you may manually compile your source code using the `astrosf77` or `astrosc` commands and then issuing the `makelocalastros` command. Since the `makelocalastros` script uses the Unix `make` program, this will avoid the compilation by the script and allow optimized code to be used.

To see the default compiler options on your local machine issue the following:

```
astrosf77 or astrosc
```

These scripts are self documenting when no arguments are given.

5.2.4 The Online Manuals

The entire suite of **ASTROS** manuals is available online in the Adobe Portable Document Format (PDF). This allows you to view the documentation on any computer that has the Adobe[®] Acrobat[®] Reader 3.0. If available from Adobe, the reader for your host computer, or computers, was delivered with your system. Any other readers that become available can be downloaded from the Adobe Web site at www.adobe.com.

To use the documents, from the command line you enter:

```
uaidoc [manual_name]
```

If you omit the `manual_name`, then you will see a splash screen that allows you to navigate to the appropriate manual. You may also go directly to a manual by placing its name on the command lines. The names of the **ASTROS** manuals are:

- `astros_theory` (Theoretical Manual)
- `astros_ref` (User's Manual)
- `astros_prog` (Programmer's Manual)
- `astros_schema` (**eBase** Schemata Manual)
- `system_support_unix` (Installation and System Support Manual)

Check the UAI Web site at www.uai.com for any interim updates and additions to the electronic documentation.

5.3 DYNAMIC MEMORY MANAGEMENT

The architecture of **ASTROS** allows the analysis of finite element models of virtually unlimited size. Most numerical calculations perform at maximum efficiency when all data for the operation fits in the working memory space of the program. Many operations may be performed even when all data that they require do not fit in memory by using what is called **spill logic**. Spill logic simply involves the paging of data to and from disk storage devices as necessary. For very large jobs, spill commonly occurs. In such cases, providing **ASTROS** with additional memory can often improve

performance. This is especially true when using the high-performance sparse matrix solvers. For these algorithms, maximum memory is important for optimal performance.

Some performance problems can develop on virtual memory machines where the working memory used by **ASTROS** exceeds the physical memory available in the machine. This can happen quite often with the high-performance sparse solvers that have large working memory requirements. Certain matrix algorithms, in particular some of the **MPYAD** and **FBS** methods, will cause excessive system paging which will degrade system performance. In these cases the user can use the **PHYSICAL** option on the **MEMORY** Resource Control Command or the configuration parameter `Physical-Memory` to restrict these algorithms to only use a portion of the working memory. This memory restriction will only be used for those algorithms that have shown to cause excessive paging. Other methods, such as the high-performance sparse solvers that function well in a virtual memory environment, will not be limited. The exact value to use on these options must be determined by the user because it is affected by the amount of physical memory available to your jobs and the current load on the system.

The working memory for **ASTROS** is dynamically acquired during execution. The amount of space that is actually used by the program is determined, in order of precedence, by the **MEMORY** Resource Control Command, the `-m` option of the `astros` script, and the configuration parameter `Working-Memory`.

Some Unix machines require special actions when it is necessary to execute **ASTROS** with very large working memory values. These actions may require rebuilding the Unix kernel, or changing the UAI software executables themselves. Please contact UAI if you experience problems running very large jobs.

5.3.1 Sparse Matrix Solvers

ASTROS has two high-performance solvers which take advantage of the latest developments in sparse matrix algorithm technology. The first of these is the symmetric matrix decomposition used in static analyses, and the second is the Lanczos eigen extraction method. This latter method is used for extracting a modest number of eigenvalues from very large systems. When these solvers are used, memory requirements may become significant. The figures below give upper and lower bound estimates for the amount of memory that users should specify on their **MEMORY** Resource Control Command. Note that if users do not specify enough memory for the new algorithms, the program will revert to the standard solution algorithms. Also note that these curves have been created using a representative sample of real analysis jobs. They are intended only to be used as guidelines — a specific job may take significantly more or less memory than indicated. On virtual memory machines, you should not be concerned if the required working memory exceeds the actual physical memory installed on the machine. These solvers have been optimized to run efficiently in this environment.

5.4 ASTROS PREFERENCE FILES

As discussed in Chapter 2, **eBase** provides System, User and Local Preference Files that can be used to override the parameters in the Default Preference File. In **ASTROS**, the **-ps**, **-pu**, and **-pl** options of the **astros** script are used to request this feature. These value are the substitution strings used by **eBase** to generate the actual file names. If these options are not used, then **eBase** will search for System, User, and Local override Preference Files having the name **uai.pref**. First the installation directory is searched, then the user's home directory, and finally, the current working directory. The names of these files, and their location, may be modified for your site by changing the Default Preference Files parameters in the **Host** section as described in Chapter 3.

5.5 ASTROS OVERRIDES OF eBase KERNEL CONFIGURATION

The **ASTROS** I/O subsystem is based on the **eBase** Kernel described in Chapter 3. There are two kinds of **ASTROS** databases. The first is the database used to store transient data during execution. This is called the **Run-Time Database**, **RUNDB**. The second kind of database is the **System Database** used by **ASTROS** to store internal data. This database is delivered with the system and is only used in a Read-Only access mode by the program.

5.5.1 The ASTROS Interface to eBase

The **ASTROS** user interface to databases is simple. As shown in the **ASTROS** User's Manual, the **ASSIGN** command is used to define databases:

```
ASSIGN logical_name [ = phys_name ] [ { NEW
                                     OLD
                                     TEMP } ] , USE = { RUNDB
                                                         ARCHIVE }
[ ,REALLOC ][ ,PASSWORD = pass]
```

Actually, there are additional parameters that are available to users when assigning database files. The complete general form of the command:

```
ASSIGN logical_name [ = phys_name ] [ { NEW
                                     OLD
                                     TEMP } ] , USE = { RUNDB
                                                         ARCHIVE }
[ ,REALLOC ][ ,PASSWORD = pass]
[ ,IBLKSIZE = int_val ][ ,ILOC = index_dir]
[ ,DBLKSIZE = int_val ][ ,DLOC = db_dir_list]
```

Each of these command parameters is used to override the **eBase** kernel configuration parameters as described in the following sections.

The **db_name** parameter has one of the following formats:

```
logical_name
logical_name = phys_name
logical_name = phys_loc/phys_name
logical_name = phys_loc/
```

5.5.2 Case Sensitivity of the ASSIGN Command

Previous versions of *ASTROS* automatically converted all input data to upper case including any Executive Control **ASSIGN** and **INCLUDE** commands. Now, the file name portion of any **ASSIGN** and **INCLUDE** commands will not be converted to upper case. All other portions of the command are case insensitive with the exception of the logical name. For the purposes of matching logical names in the data stream, logical names are also case insensitive. But if the logical name is used to generate the physical file name, because it is omitted, then the actual case entered is used.

5.5.3 Overriding an eBase Location

The directory where database files are stored comes from one of three places: the directory information in the *phys_loc* field of the *db_name* parameter; the **ILOC** and **DLOC** parameters; or the **eBase** Configuration Section. These methods apply to both Temporary and Permanent databases.

The directory information in the *phys_loc* field provides a convenient method to place a database when it is desired to have both the Index and Data Component in the same directory and only one Data Component file is required. It is illegal to use this method if you have selected either an **ILOC** or **DLOC** parameter. The parameter:

```
ILOC=path_loc
```

specifies the full path name of a directory where you want to store the Index Component File of the database being **ASSIGNED**. This applies whether the database exists or is being created. Similarly, either form of the parameter:

```
DLOC=path_loc
DLOC=(path_loc1,path_loc2,...)
```

specifies the full path name of one or more directories where you want to store the Data Component Files of the database being **ASSIGNED**.

If no directory information is provided with the *phys_loc* field, and neither **ILOC** nor **DLOC** are specified, then the **eBase** Configuration parameters discussed in Chapter 3 are used.

For example, to **ASSIGN** a temporary **RUNDB** Database whose Index and Data Components will be placed in the directory `/usr1/tmp`:

```
ASSIGN RUNDB=/usr1/tmp/,TEMP,USE=RUNDB
```

When your users are using or generating large databases, they may also distribute the data across multiple file systems. Consider the example to **ASSIGN** a Permanent **RUNDB** Database whose Index Component will reside in `/usr1/tmp` and whose Data Components will reside in `/usr2/tmp`, `/usr3/tmp`, and `/usr4/tmp`:

```

ASSIGN RUNDB=keeprun,NEW,PASSWORD=pass,
      ILOC=/usr1/tmp,
      DLOC=(/usr2/tmp,/usr3/tmp,/usr4/tmp)

```

Note how the **USE** value will default to the logical name (in this case **RUNDB**) as a further convenience.

5.5.4 Creating the *eBase* Name

The **ASTROS** databases follow the general naming conventions described in Chapter 3. The naming conventions for the physical files differ for Permanent and Temporary databases. This is illustrated in the following table.

eBase FILE NAMES GENERATED BY VARIOUS FORMS OF <i>db_name</i>	
<i>logical_name=/phys_loc/phys_name</i> or <i>logical_name=phys_name</i>	<i>logical_name</i> or <i>logical_name=/phys_loc/</i>
FOR PERMANENT DATABASES	
<i>phys_name.*</i>	<i>logical_name.*</i>
FOR TEMPORARY DATABASES	
<i>phys_namexxxxx.*</i>	<i>logical_namexxxxx.*</i>
if <i>phys_name</i> or <i>logical_name</i> is longer than six characters <i>uniquexxxxx.*</i>	
The notation ".*" indicates that all of the database files, both index and data, have the specified name.	

As shown, to specify a name, the *db_name* parameter of the **ASSIGN** command is used. In its simplest form, this parameter gives just the Logical Name of the **eBase** database and this name is also used to generate the physical file names. If the *db_name* parameter also contains a *phys_name* field then this name is used instead. The general form of the *db_name* parameter was discussed in a previous section of this chapter.

The name generation for temporary databases follows the same general rules as for permanent databases except the **eBase** kernel will also append the five character Unix process id on the end of the name. This is to insure that the temporary file names will be unique for all jobs running simultaneous on the machine. Also to insure that the temporary file names are valid on all versions of Unix the total file name will not exceed fourteen characters. Therefore the *logical_name* or *phys_name* field used should not contain more than six characters. If it does, then **eBase** will automatically generate a unique six character name to use instead.

You can also use the asterisk (*) character in the *phys_name* portion of the *db_name* parameter. In this case, the asterisk is replaced by the name of the input data file with any file extensions removed.

5.5.5 Changing *eBase* Block Size Parameters

Database block sizes are also controlled by the *eBase* Configuration Section or the `DBLKSIZ` and `IBLKSIZ` parameters. The two `ASSIGN` command parameters:

```
IBLKSIZ=int_val
DBLKSIZ=int_val
```

are used to control the size of physical blocks used for performing I/O operations. The first, `IBLKSIZ`, specifies the block size for the Index Component, and the second parameter, `DBLKSIZ`, specifies the block size of the Data Component. All block size values are specified in single precision words. For example, to `ASSIGN` a temporary `RUNDB` whose Data Component block size is 8192 words:

```
ASSIGN MYRUNDB, TEMP, USE=RUNDB, DBLKSIZ=8192
```

While there are no restrictions on the block sizes of the various databases `ASSIGNED` during a *ASTROS* execution, when moving data among more than one database, it is more efficient if the databases have the same Data Component block sizes.

5.5.6 Assigning an Old *eBase* in *ASTROS*

To `ASSIGN` a database that was created in another *ASTROS* execution, the disposition of `OLD` is used. It is not necessary to respecify all of the characteristics that were used when the database was created, because this information is saved on the database. These characteristics include blocks sizes and directory paths for the data components of the database. The only parameters that must be specified are the physical name, the directory path to the index file, and the correct password. This is illustrated in the following example:

When the database is created:

```
ASSIGN ARCHIVE, NEW, PASSWORD=pass,
      DLOC=( /usr1/tmp. /usr2/tmp ),
      IBLKSIZ=256, DBLKSIZ=8192
```

When the database is reused:

```
ASSIGN ARCHIVE, OLD, PASSWORD=pass
```

5.6 THE *ASTROS* CONFIGURATION SECTION

The *ASTROS* Configuration Section, whose parameters define default values for system and engineering data which are unique to *ASTROS*, is described in Section A.5.

This page is intentionally blank.

6. APPLICATION PROGRAMMING INTERFACES: *eBase:applib and matlib*

6.1 AUDIENCE

This Chapter is intended for use by the person or group responsible for supporting any **eBase** application created by your organization. It provides you with information describing software resource utilization so that you may make changes to the **eBase** applications to tune the software for your computing environment and to set them up in a manner that is familiar to your end-users. It also describes the additional libraries delivered with the **eBase:applib** and **eBase:matlib** products.

6.2 USING THE **eBase** SOFTWARE SUITE

This Chapter describes the manner in which you use the **eBase:applib** and **eBase:matlib** Application Programming Interfaces in developing software applications.

6.2.1 The Online Manuals

The application interface manuals are available online in the Adobe Portable Document Format (PDF). This allows you to view the documentation on any computer that has the Adobe[®] Acrobat[®] Reader 3.0. If available from Adobe, the reader for your host computer, or computers, was delivered with your system. Any other readers that become available can be downloaded from the Adobe Web site at www.adobe.com.

To use the documents, from the command line you enter:

```
uaidoc [manual_name]
```

If you omit the *manual_name*, then you will see a splash screen that allows you to navigate to the appropriate manual. You may also go directly to a manual by placing its name on the command lines. The names of the **eBase:applib** and **eBase:matlib** manuals are:

- eBase_applib**
- eBase_matlib**

Check the UAI Web site at www.uai.com for any interim updates and additions to the electronic documentation.

6.3 USING THE APPLICATIONS PROGRAMMING INTERFACES

6.3.1 Application Development

The **eBase:applib** and **matlib** subroutine libraries, simply called the **API** in the remainder of this Chapter, are contained in four archive files:

<code>rel_dir/applib#/applib.a</code>	— eBase:applib Library
<code>rel_dir/applib#/matlib.a</code>	— eBase:matlib Library
<code>rel_dir/applib#/blas.a</code>	— BLAS Library
<code>rel_dir/applib#/lapack.a</code>	— LAPACK Library

The **BLAS** and **LAPACK** libraries, used by some **eBase:matlib** routines, are delivered as separate libraries so that they may be replaced by any custom optimized versions that may be delivered with your computer.

Which libraries are used, and the proper link order, depends on the type of application you are developing, as shown in the following table:

eBase:applib Application
<code>rel_dir/applib#/applib.a</code> custom system libraries
eBase:matlib Application
<code>rel_dir/applib#/matlib.a</code> custom optimized versions of LAPACK and BLAS <code>rel_dir/applib#/lapack.a</code> <code>rel_dir/applib#/blas.a</code> <code>rel_dir/applib#/applib.a</code> custom system libraries

The general command line for compiling an **applib** or **matlib** program is:

```
compiler -o prog custom_compiler_opts obj1.f obj2.f ... LibList
```

The following table shows special compiler options, or libraries, that are required for each host computer.

HOST COMPUTER	<i>compiler</i>	<i>custom compiler options</i>	<i>custom LAPACK/BLAS</i>	<i>custom system libraries</i>
IBM RS/6000	f77	none	-lblas	none
Cray	cf77	none	none	none
DEC Alpha	f77	none	-ldxml	-lbsd
HP 9000/700	f77	none	-lvec -lblas	none
SUN	f77	none	none	-lV77
SGI R4000	f77	-non_shared	none	none
SGI R8000	f77	none	none	none
OTHERS	f77	none	none	none

In sophisticated development environments, the libraries are simply added to the search list in all appropriate locations.

When executing your **API** application, you must ensure that the routines can find and read the Default Preference File. This is done by setting the **UAICONFIG** environment variable to point to this file. If you use a script which invokes your **API** application, you may set the **UAICONFIG** variable in the script before invoking the application. More generally, you may set **UAICONFIG** variable in your shell initialization file, so it exists for any **API** application which you run. In a similar manner, the routines must be able to read the User Verification File, **uaiuvf**. This is done by setting the **UAIUVF** environment variable to point to the file.

To set the **UAICONFIG** and **UAIUVF** environment variables in a shell compatible with the Bourne Shell, use:

```
UAICONFIG=rel_dir/applib#/applib.pref
export UAICONFIG
UAIUVF=rel_dir/uaiuvf
export UAIUVF
```

To set them in a shell compatible with the C-Shell, use:

```
setenv UAICONFIG rel_dir/applib#/applib.pref
setenv UAIUVF rel_dir/uaiuvf
```

If you use some other shell which is not compatible with either of the above, see its documentation on how environment variables are set.

6.3.2 THE *eBase:applib* PROGRAMMING EXAMPLES

The *applib* programming examples may be found in the directory:

```
rel_dir/applib#/applib
```

This directory consists of four Fortran source code files that show examples of *applib* usage. Data files for these programs are also included.

The `sebgen` program generates a test *eBase* database which is used to perform the examples shown in the *eShell User's Manual*. This program requires the `SEBGEN.*` data files to execute.

The `export` program demonstrates how the *applib* routines can be used to export an *eBase* database to a file. This sample program prompts you for several parameters. The file `export.input` contains appropriate responses to these prompts and can be used as standard input to the program.

The `import` program demonstrates the use of *applib* utilities to import the file generated by the export program and create an *eBase* database. A sample input file is also provided for this program.

The `applibver` is used to show the Release and Version of your *applib* libraries.

6.3.3 THE *eBase:matlib* PROGRAMMING EXAMPLES

The *eBase:matlib* programming examples may be found in the directory:

```
rel_dir/applib#/matlib
```

This directory contains a complete Fortran program, `sample.f`, which illustrates the usage of the *matlib* utilities.

6.3.4 Override Preference File Selection

An *applib* application can specify its own Preference File substitution string by using subroutine `EBPREF`:

```
CALL EBPREF(system_string,user_string,local_string,IRET)
```

This call should be made before any other *applib* call, otherwise strings are not set, and an error code is returned.

6.4 API OVERRIDES OF *eBase* KERNEL CONFIGURATION

This section describes how the kernel configuration parameters may be overridden in *eBase:applib*.

6.4.1 The *applib* Interface to *eBase*

The user interface for defining database names in the *eBase:applib* interface depends on the status of the database. For temporary databases, you use:

```
CALL EBTEMP(db_name,op_sys,IRET)
```

For old databases you use:

```
CALL EBOLD(db_name,PASS,ACCESS,op_sys,IRET)
```

and for new databases you use:

```
CALL EBNEW(db_name ,PWR ,PWW ,PWA ,op_sys , IRET)
```

In all cases, the *db_name* parameter has one of the following formats:

```
logical_name
logical_name = phys_name
logical_name = phys_loc/phys_name
logical_name = phys_loc/
```

The *op_sys* parameter is a list of host computer-dependent options. Some of these override Configuration parameters and others provide file locations and dispositions. The Unix options are shown in the following table.

PARAMETER AND FORMAT	DESCRIPTION
ILOC = <i>path_loc</i>	Specifies the location of the Index Component file.
DLOC = <i>path_loc</i> DLOC = (<i>path_loc1</i> , <i>path_loc2</i> ,...)	Specifies the location of the Data Component files.
IBLKSIZE = <i>int_val</i>	Specifies the Index Component block size.
DBLKSIZE = <i>int_val</i>	Specifies the Data Component block size.
INTEGRITY = { HIGH LOW }	Specifies the integrity level for the database.
REALLOCATE	Specifies that if the database already exists, it is deleted and reused.

Multi options may be included by separating them with blanks or commas. The string of options must be enclosed in single quotation marks.

6.4.2 Overriding an *eBase* Location

The directory where database files are stored comes from one of three places: the directory information in the *phys_loc* field of the *db_name* parameter; the ILOC and DLOC parameters; or the *eBase* Configuration Section. These methods apply to both Temporary and Permanent databases.

The directory information in the *phys_loc* field provides a convenient method to place a database when it is desired to have both the Index and Data Component in the same directory and only one Data Component file is required. It is illegal to use this method if you have selected either an ILOC or DLOC parameter. The parameter:

```
ILOC=path_loc
```

specifies the full path name of a directory where you want to store the Index Component File of the database. This applies whether the database exists or is being created. Similarly, either form of the parameter:

```
DLOC=path_loc
DLOC=(path_loc1,path_loc2,...)
```

specifies the full path name of one or more directories to hold the Data Component Files of the database.

If no directory information is provided with the *phys_loc* field, and neither *ILOC* nor *DLOC* are specified, then the **eBase** Configuration parameters discussed in Chapter 3 are used.

For example, to create a new Database named *myeb* whose index and Data Components will be placed in the directory */usr1/tmp*:

```
CALL EBNEW('myeb=/usr1/tmp/' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' IRET)
```

Consider the example to create a Permanent Database, *perm*, whose Index Component will reside in */usr1/tmp* and whose data components will reside in */usr2/tmp*, */usr3/tmp*, and */usr4/tmp*:

```
CALL EBNEW('perm' ' ' ' ' ' ' ' ' ' ' ILOC=/usr1/tmp
            DLOC=(/usr2/tmp, /usr3/tmp, /usr4/tmp) ' IRET)
```

6.4.3 Creating the *eBase* Name

Databases follow the general naming conventions described in Chapter 3. The naming conventions for the physical files differ for Permanent and Temporary databases. This is illustrated in the following table.

eBase FILE NAMES GENERATED BY VARIOUS FORMS OF <i>db_name</i>	
<i>logical_name=/phys_loc/phys_name</i> or <i>logical_name=phys_name</i>	<i>logical_name</i> or <i>logical_name=/phys_loc/</i>
FOR PERMANENT DATABASES	
<i>phys_name.*</i>	<i>logical_name.*</i>
FOR TEMPORARY DATABASES	
<i>phys_name#xx.*</i>	<i>logical_name#xx.*</i>
if <i>phys_name</i> or <i>logical_name</i> is longer than six characters <i>unique#xx.*</i>	
The notation ".*" indicates that all of the database files, both index and data, have the specified name.	

As shown, to specify a name, the *db_name* parameter of the **API** calls is used. In its simplest form, this parameter gives just the Logical Name of the **eBase** database and this name is also used to generate the physical file names. If the *db_name* parameter also contains a *phys_name* field then this name is used instead. The general form of the *db_name* parameter was discussed in a previous section of this chapter.

For example, to create a new database named *myeb* in the current working directory, which is */usr1*, use:

```
CALL EBNEW('myeb', ' ', ' ', ' ', ' ', ' ', ' ', IRET)
```

which creates the physical files¹:

```
/usr1/myeb.edb and
/usr1/myeb.00
```

To understand the use of *phys_name*, consider an example to create a new database whose logical name is *myeb*. Store this in the physical file *new.ebase* again assuming the working directory is */usr1*:

```
CALL EBNEW('myeb=new.ebase', ' ', ' ', ' ', ' ', ' ', ' ', IRET)
```

which creates the physical files (as above):

```
/usr1/new.ebase.edb and
/usr1/new.ebase.00
```

The name generation for temporary databases follows the same general rules as for permanent databases except the **eBase** kernel will also append the five character Unix process id on the end of the name. This is to insure that the temporary file names will be unique for all jobs running simultaneous on the machine. Also to insure that the temporary file names are valid on all versions of Unix the total file name will not exceed fourteen characters. Therefore the *logical_name* or *phys_name* field used should not contain more than six characters. If it does, then **eBase** will automatically generate a unique six character name to use instead.

The actual case of the file names generated will be the exact same as those entered. This is true whether the name is derived from the *logical_name* or the *phys_name* parameter.

6.4.4 Changing eBase Block Size Parameters

Both the Index File block size and the Data Component File block size may be modified when using **eBase:applib** by using the special parameters **IBLKSIZ** and **DBLKSIZ**. For example, to create a temporary **eBase** database named *RUNDB* whose Data Component block size is **8192** single precision words:

```
CALL EBTEMP('RUNDB', 'DBLKSIZ=8192', IRET)
```

¹ Note that the precise number of Data Files created depends on the number of locations specified in the **eBase** kernel Configuration Section with the *Perm-eBase-Data-Loc* parameter. Similarly, the examples assume that the index and data locations specified by the *Perm-eBase-Index-Loc* and *Perm-eBase-Data-Loc* parameters are '.', which is the current working directory.

6.4.5 Changing Other *eBase* Kernel Parameters

The integrity of databases may also be changed in *eBase:applib* by using the **INTEGRITY** option. To create a new permanent *eBase* database named **myeb** and set its integrity maintenance to **HIGH**:

```
CALL EBNEW('myeb',' ',' ',' ',' ','INTEGRITY=HIGH',IRET)
```

In addition to these, you may also specify the keyword **REALLOCATE**. When this option is specified, and it is only used for databases which are **NEW**, *eBase* will delete a previously existing database of the same name and replace it with the new one.

6.4.6 Using an Old *eBase* in the API

When you attach an **OLD** database to your **API** application, it is not necessary to respecify all of the characteristics that were used when the database was created because they are stored on the database. These include the block sizes and directory paths for the data files comprising the database. All that is necessary is to specify the physical name, the directory path to the index file, and the correct password. This is illustrated in the following example:

When the database is created:

```
CALL EBNEW('myeb=new.ebase',' ',' ','password',
          'DLOC=(/usr1/tmp,/usr2/tmp',
          IBLKSIZE=256,DBLKSIZE=8192,' ',IRET)
```

When the database is reused:

```
CALL EBOLD('myeb=new.ebase','password','READ',' ',IRET)
```

6.5 DYNAMIC MEMORY

The architecture of *eBase* allows for handling databases of virtually unlimited size. This is facilitated through the use of Dynamic Memory. *eBase* applications have two pools of dynamic memory. The first is used by the *eBase* kernel to perform database operations and is controlled by parameters explained in Chapter 3. The second, used by an application for working memory, is described below.

6.5.1 Application Dynamic Memory

Applications using *eBase:applib* may use the dynamic memory feature to create more flexible software. By using dynamic memory, the programmer can avoid using fixed length arrays. This means that algorithms may be coded to function properly for virtually unlimited size problems.

The programmer calls the Dynamic Memory Management System (**DMMS**) to get memory. In turn, the Dynamic Memory Management System allocates memory from Unix to satisfy programmer requests, according to three configuration parameters:

Dynamic-Initial-Memory
Dynamic-Memory-Increment
Dynamic-Max-Memory

Unlike the **eBase** kernel memory, the programmer may override these parameters by calling **DMSIZE**.

6.5.2 **eBase:matlib**

Most of the **eBase:matlib** subroutines include an argument for an array of working memory. You must perform all appropriate memory allocations with the **eBase:applib** utilities before invoking the Matrix operations. Note that the high performance sparse matrix utilities have memory requirements which are shown in Chapter 4.

6.6 THE **eBase:applib** and **eBase:matlib** CONFIGURATION SECTIONS

The configuration parameters relating to the **eBase:applib** and **eBase:matlib** utility libraries are presented in Section A.6 and Section A.7.

This page is intentionally blank.

7. THE *eShell* PROGRAM

7.1 AUDIENCE

This Chapter is intended for use by the person or group responsible for supporting the *eShell* interactive interface program within your organization. It provides you with information describing software resource utilization so that you may make changes to the *eBase* procedures to tune the program for your computing environment and to set up the suite in a manner that is familiar to your end-users. It also describes the additional libraries delivered with the program.

7.2 USING THE *eShell* PROGRAM

This section describes the manner in which you execute *eShell*, the *eBase* Interactive Interface.

7.2.1 The *eShell* Program

To execute *eShell* you enter:

```
eshell [-ps prefname] [-pu prefname]  
        [-pl prefname] [database]
```

where:

prefname Specifies the substitution string used to generate Preference File names. You may specify a different string for the System (*-ps*), the User (*-pu*) and the Local (*-pl*) preference files. If you have the unusual case where all of these files have the same name, you may use the option *-p* followed by the *prefname*.

database Is the name of a database to be opened with read access.

This script will execute **eShell** in the interactive mode. Unless directed otherwise by **eShell** commands, all subsequent output will be sent to the terminal device.

7.2.2 The Tutorial Examples

The **eShell** Tutorial Problem library may be found in the directory:

```
rel_dir/eshell#/tutorial
```

It consists of a set of Script Files which contain the commands used in each of the Examples presented in the **eShell User's Manual**. Each file in the directory has a name of the form:

```
excc.in
```

where *cc* represents the Chapter number, i.e. Chapter 1 is 01 and Chapter 12 is 12. The resulting output is also included in this directory with file names:

```
excc.out
```

They are stored in this manner to allow them to be executed in the natural pedagogical sequence in which they were presented.

7.2.3 The Online Manuals

The **eShell** User's Manual and related manuals are available online in the Adobe Portable Document Format (PDF). This allows you to view the documentation on any computer that has the Adobe® Acrobat® Reader 3.0. If available from Adobe, the reader for your host computer, or computers, was delivered with your system. Any other readers that become available can be downloaded from the Adobe Web site at www.adobe.com.

To use this and related manuals, from the command line you enter:

```
uaidoc [manual_name]
```

If you omit the *manual_name*, then you will see a splash screen that allows you to navigate to the appropriate manual. You may also go directly to a manual by placing its name on the command lines. The names of the manuals related to **eShell** are:

- eshell*
- nastran_schema*
- astros_schema*

Check the UAI Web site at www.uai.com for any interim updates and additions to the electronic documentation.

7.3 *eShell* PREFERENCE FILES

As discussed in Chapter 2, *eBase* provides System, User and Local Preference Files that can be used to override the parameters in the Default Preference File. In *eShell*, the `-ps`, `-pu`, and `-pl` options of the `eshe11` script are used to request this feature. These value are the substitution strings used by *eBase* to generate the actual file names. If these options are not used, then *eBase* will search for System, User, and Local override Preference Files having the name `uai.pref`. First the installation directory is searched, then the user's home directory, and finally, the current working directory. The names of these files, and their location, may be modified for your site by changing the Default Preference Files parameters in the `Host` section as described in Chapter 3.

7.4 *eShell* OVERRIDES OF KERNEL CONFIGURATION

This section describes the manner in which the *eShell* application overrides the basic *eBase* kernel configuration parameters described in Chapter 3.

7.4.1 The *eShell* Interface to *eBase*

The user interface for defining database names in *eShell* is the command:

$$\text{OPEN } db_name \left\{ \begin{array}{l} \text{NEW} \\ \text{TEMP} \\ \text{WITH } \left\{ \begin{array}{l} \text{READ} \\ \text{WRITE} \\ \text{ADMIN} \end{array} \right\} \end{array} \right\} ['op_sys'] ;$$

This command is used to either create a new or temporary database or to open an existing database for use.

Where the `db_name` parameter has one of the following formats:

```
logical_name
logical_name = phys_name
logical_name = phys_loc/phys_name
logical_name = phys_loc/
```

In all *eShell* commands, the `phys_name` and `phys_loc` pair may be enclosed in single quotation marks or not. Unquoted names, however, are limited to 32 characters and cannot have any special characters in them except '\$' and '_'.

The `op_sys` parameter is a list of host computer-dependent options. Some of these override Configuration parameters and others provide file locations and dispositions. The Unix options are shown in the following table.

PARAMETER AND FORMAT	DESCRIPTION
<code>ILOC = path_loc</code>	Specifies the location of the Index Component file
<code>DLOC = path_loc</code> <code>DLOC = (path_loc1,path_loc2,...)</code>	Specifies the location of the Data Component files.
<code>IBLKSIZE = int_val</code>	Specifies the Index Component block size.
<code>DBLKSIZE = int_val</code>	Specifies the Data Component block size.
<code>INTEGRITY = { HIGH } { LOW }</code>	Specifies the integrity level for the database.
<code>REALLOCATE</code>	Specifies that if the database already exists, it is deleted and reused.

Multiple options may be included by separating them with blanks or commas. The string of options must be enclosed in single quotation marks.

7.4.2 Overriding an *eBase* Location

The directory where database files are stored comes from one of three places: the directory information in the *phys_loc* field of the *db_name* parameter; the **ILOC** and **DLOC** parameters; or the *eBase* Configuration Section. These methods apply to both Temporary and Permanent databases.

The directory information in the *phys_loc* field provides a convenient method to place a database when it is desired to have both the Index and Data Component in the same directory and only one Data Component file is required. It is illegal to use this method if you have selected either an **ILOC** or **DLOC** parameter. The parameter:

```
ILOC=path_loc
```

specifies the full path name of a directory where you want to store the Index Component File of the database. This applies whether the database exists or is being created. Similarly, either form of the parameter:

```
DLOC=path_loc  
DLOC=(path_loc1,path_loc2,...)
```

specifies the full path name of one or more directories to hold the Data Component Files of the database.

If no directory information is provided with the *phys_loc* field, and neither **ILOC** nor **DLOC** are specified, then the *eBase* Configuration parameters discussed in Chapter 3 are used.

For example, to create a new Database named `myeb` whose index and Data Components will be placed in the directory `/usr1/tmp`:

```
open myeb='/usr1/tmp/' new;
```

Consider the example to create a Permanent Database, `perm`, whose Index Component will reside in `/usr1/tmp` and whose data components will reside in `/usr2/tmp`, `/usr3/tmp`, and `/usr4/tmp`:

```
open perm new 'iloc=/usr1/tmp
dloc=(/usr2/tmp,/usr3/tmp,/usr4/tmp)';
```

7.4.3 Creating the *eBase* Name

Databases follow the general naming conventions described in Chapter 3. The naming conventions for the physical files differ for Permanent and Temporary databases. This is illustrated in the following table.

eBase FILE NAMES GENERATED BY VARIOUS FORMS OF <i>db_name</i>	
<i>logical_name</i> =/ <i>phys_loc</i> / <i>phys_name</i> or <i>logical_name</i> = <i>phys_name</i>	<i>logical_name</i> or <i>logical_name</i> =/ <i>phys_loc</i> /
FOR PERMANENT DATABASES	
<i>phys_name</i> .*	<i>logical_name</i> .*
FOR TEMPORARY DATABASES	
<i>phys_name</i> xxxxx.*	<i>logical_name</i> xxxxx.*
if <i>phys_name</i> or <i>logical_name</i> is longer than six characters <i>unique</i> xxxxx.*	
The notation ".*" indicates that all of the database files, both index and data, have the specified name.	

As shown, to specify a name, the *db_name* parameter of the **OPEN** command is used. In its simplest form, this parameter gives just the Logical Name of the *eBase* database and this name is also used to generate the physical file names. If the *db_name* parameter also contains a *phys_name* field then this name is used instead. The general form of the *db_name* parameter was discussed in a previous section of this chapter.

For example, to create a new database named `myeb` in the current working directory, which is `/usr1`, use:

```
open myeb new;

which creates the physical files1:

/usr1/myeb.edb and
/usr1/myeb.00
```

¹ Note that the precise number of Data Files created depends on the number of locations specified in the *eBase* kernel Configuration Section with the `Perm-eBase-Data-Loc` parameter. Similarly, the examples assume that the index and data locations specified by the `Perm-eBase-Index-Loc` and `Perm-eBase-Data-Loc` parameters are '.', which is the current working directory.

The user may also specify a physical file name by using the *phys_name* portion of the *db_name* parameter. For instance, to create a new database whose logical name is *myeb*, which resides in physical file *new.ebase*, which is in the */usr1* current working directory use:

```
open myeb='new.ebase' new;
```

which creates the physical files¹:

```
/usr1/new.ebase.edb and
/usr1/new.ebase.00
```

The name generation for temporary databases follows the same general rules as for permanent databases except the **eBase** kernel will also append the five character Unix process id on the end of the name. This is to insure that the temporary file names will be unique for all jobs running simultaneous on the machine. Also to insure that the temporary file names are valid on all versions of Unix, the total file name will not exceed fourteen characters. Therefore the *logical_name* or *phys_name* field used should not contain more than six characters. If it does, then **eBase** will automatically generate a unique six character name to use instead.

The actual case of the file names generated will be the exact same as those entered. This is true whether the name is derived from the *logical_name* or the *phys_name* parameter.

7.4.4 Changing eBase Block Size Parameters

Both the Index File block size and the Data Component File block size may be modified when using **eShell** by using the parameters **IBLKSIZE** and **DBLKSIZE**. To create a temporary **eBase** database named *RUNDB* whose Data Component block size is **8192** words you use:

```
open rundb temp 'dblksize=8192';
```

7.4.5 Changing Other eBase Kernel Parameters

The integrity of databases may also be changed in **eShell** by using the **INTEGRITY** option. For example, to create a new permanent **eBase** database named *myeb* and set its integrity maintenance to **HIGH**:

```
open myeb new 'integrity=high';
```

Additionally you may specify the keyword **REALLOCATE**. When this option is specified, and it is only used for databases which are **NEW**, **eBase** will delete a previously existing database of the same name and replace it with the new one.

7.4.6 Using an Old eBase in eShell

When you attach an **OLD** database to **eShell**, it is not necessary to respecify all of the characteristics that were used when the database was created because they are stored on the database. These include the block sizes and

directory paths for the data files comprising the database. All that is necessary is to specify the physical name, the directory path to the index file, and the correct password. This is illustrated in the following example:

```

When the database is created:

open mydb new 'dloc=(/usr1/tmp,/usr2/tmp,
              iblksize=256,dblksize=8192;

When the database is reused:

open mydb old;

```

7.5 *eShell* INTERFACE FILES

eShell uses several sequential files which are called the **SCRIPT**, **INTERFACE**, **REPORT** and **EXPORT** files. These are available in *eShell* using the commands shown in the table below.

SEQUENTIAL FILE	<i>eShell</i> COMMAND
SCRIPT	SET SCRIPT TO ' <i>file_name</i> ';
	SET ARCHIVE TO ' <i>file_name</i> ';
INTERFACE	SET INTERFACE TO ' <i>file_name</i> ';
REPORT	SET REPORT TO ' <i>file_name</i> ';
EXPORT	EXPORT [<i>path</i>] ' <i>file_name</i> ';
	IMPORT ' <i>file_name</i> ' [<i>path</i>];

7.6 *eShell* DYNAMIC MEMORY

eShell is simply a software application which uses the **eBase:applib** Applications Programming Interface. It therefore uses Application Dynamic Memory. Other applications using **eBase:applib** may use the dynamic memory feature to create more flexible software. By using dynamic memory, the programmer can avoid using fixed length arrays. This means that algorithms may be coded to function properly for virtually unlimited size problems. In the case of *eShell*, there are three Configuration parameters,

```

Initial-Memory
Memory-Increment
Max-Memory

```

which are used for this purpose. These parameters override the three dynamic memory parameters in the **applib** configuration section for *eShell*.

7.7 THE *eShell* CONFIGURATION SECTION

The *eShell* Configuration parameters are described in Section A.8.

This page is intentionally blank.

8. UAI/NASTRAN GRAPHICS SOFTWARE

8.1 AUDIENCE

UAI/NASTRAN has extensive plotting capabilities which include displays of the structural model and X-Y graphs. These features are described in detail in the **UAI/NASTRAN** User's Guide and the **UAI/NASTRAN** User's Reference Manual. The plotting software creates either a binary or a formatted plot file that may be interfaced to any hardware graphics device by writing an appropriate post-processing program.

This section describes the plotting programs provided by UAI to process the plot files which may be created by **UAI/NASTRAN** executions. Also discussed is the format of these files and the representation of the data within them so that you may modify and enhance the software provided or write your own for other purposes.

8.2 THE PLOTTING PROGRAMS

Four plotting programs, `tekplot`, `nastplotps`, `nastplotgl` and `nastplot` are provided. Each of these programs may be used to create plot displays on graphics terminals or popular hardcopy devices or create new files using either the PostScript, HP-PCL or HP-GL languages. Additionally, source code is provided in the form of program `tekplot` which provides you a starting point for creating your own customized plotting program.

8.2.1 The Tektronix PLOT10 Plot Program

A Fortran program, `tekplot`, is provided in source code format, which you may modify and use to process **UAI/NASTRAN** plot files and create displays on graphics terminals connected to your host computer which support the Tektronix PLOT10 graphics instructions. Note, that in order to use this program, you will have to compile the source and then link the object code with a PLOT10 compatible library which you provide. The `tekplot` source code may be found in:

```
rel_dir/nastran#/utility/tekplot.f
```

The `tekplot` source code is also an ideal starting point to generate a plot program to support other, special hardware plotting or drawing devices which are at your location. This source code is completely self documenting with comment statements, and you should have no difficulty in making changes to support other devices such as pen plotters.

8.2.2 The PostScript Plot Program

The program `nastplotps`, included with this delivery, will read both binary and formatted plot files generated by **UAI/NASTRAN** and generate an Encapsulated PostScript file. This PostScript output can then be either sent to a printer or imported into a text formatting program which accepts Encapsulated PostScript input. The program allows the user to select plots, select fonts, control paper size and to determine output orientation (landscape or portrait). Detailed documentation on these options is available by executing the following command with no arguments:

```
nastplotps
```

The on-line help generated is as follows:

```
Usage: nastplotps options file_name_1 file_name_2 ...

Options supported are:

    -b = plot files are binary (default)
    -f = plot files are formatted
    -nf = suppress frame around plot
    -pn# = only plot number # is processed
    -pw# = paper width (default -pw8.5)
    -mw# = unplottable margin width (default -mw0.25)
    -ph# = paper height (default -ph11.0)
    -mh# = unplottable margin height (default -mh0.25)
    -pro = profile orientation (default)
    -lan = landscape orientation
    -tx = typeface (default -tHelvetica)
```

The output of **nastplotps** is to Unix standard output. Normally, you should redirect standard output to a file or pipe it to a print spooling program as desired. The following illustrates a typical use of **nastplotps**:

```
nastplotps -f -lan mydata.plt | lpr -Pps
```

8.2.3 The HP-GL Plot Program

The program **nastplotgl**, included with this delivery, will read both binary and formatted plot files generated by **UAI/NASTRAN** and generate plots using the HP-GL language. This HP-GL output can then be either sent to a printer or imported into a text formatting program which accepts HP-GL input. The program allows the user to select plots and control paper size. Detailed documentation on these options is available by executing the following command with no arguments:

```
nastplotgl
```

The on-line help generated is as follows:

```
Usage: nastplotgl options file_name_1 file_name_2 ...

Options supported are:

    -b = plot files are binary (default)
    -f = plot files are formatted
    -nf = suppress frame around plot
    -pn# = only plot number # is processed
    -pw# = paper width (default -pw8.5)
    -mw# = unplottable margin width (default -mw0.25)
    -ph# = paper height (default -ph11.0)
    -mh# = unplottable margin height (default -mh0.25)
```

The output of **nastplotgl** is to Unix standard output. Normally, you should redirect standard output to a file or pipe it to a print spooling program as desired. The following illustrates a typical use of **nastplotgl**:

```
nastplotgl -f mydata.plt | lpr -Pgl
```

8.2.4 The X Window System, Motif Interface Plot Program

For computer systems which support the X Window System, the plotting program `nastplot` is provided. This program operates in the X environment, and it uses a Motif interactive interface. `nastplot` provides the following functional capability for viewing and processing **UAI/NASTRAN** plot files:

- Automatic recognition and processing of binary or formatted plot files.
- Full support of the **UAI/NASTRAN** `LINestyle` command using user selectable display colors.
- Direct selection of display for any plot in the plot file.
- Zooming of the plot display.
- Export of plots to either a printer or a file, using either PostScript or HP PCL display languages.

As with most Motif applications, you have complete control over the appearance of the program using resources. This includes control over both fonts and colors used for different portions of the display. The resources define up to 10 user configurable printers. This includes print format, paper size and orientation, and font selection. You may also specify the desired operating system command which will send the output to the desired hardcopy device.

The following table presents a description of the resource entries supported by `nastplot` and used for controlling printing of data displayed when using `nastplot`.

RESOURCE	DEFAULT VALUE and/or CHOICES	DESCRIPTION
PrintLabel#: <i>label</i>	NONE	Label for selection in the selection box.
PrintCommand#: <i>command</i>	NONE	Command to process the Plot output. If undefined the output will be put in a file of the users choice. The command must include the special symbol %s which will be substituted with a temp file name which contains the plot data.
PrintFormat#: <i>string</i>	{ PCL PostScript }	Format for the print output.
PrintFrame#: <i>string</i>	{ TRUE FALSE }	If TRUE a frame is printed around the page.
PrintPaperx#: <i>integer</i>	{ 8.0(PCL) 8.5(PostScript) x.x }	The width of the paper in inches.

RESOURCE	DEFAULT VALUE and/or CHOICES	DESCRIPTION
PrintMargx#: <i>integer</i>	$\left\{ \begin{array}{l} 0.0(\text{PCL}) \\ 0.25(\text{PostScript}) \\ x.x \end{array} \right\}$	The width of the margin in the horizontal direction in inches.
PrintPapery#: <i>integer</i>	$\left\{ \begin{array}{l} 10.6(\text{PCL}) \\ 11.0(\text{PostScript}) \\ x.x \end{array} \right\}$	The height of the paper in inches.
PrintMargy#: <i>integer</i>	$\left\{ \begin{array}{l} 0.0(\text{PCL}) \\ 0.25(\text{PostScript}) \\ x.x \end{array} \right\}$	The height of the margin in the vertical direction in inches.
PrintOrient#: <i>string</i>	$\left\{ \begin{array}{l} \text{PORTRAIT} \\ \text{LANDSCAPE} \end{array} \right\}$	The orientation of the page.
PrintFont#: <i>string</i>	$\left\{ \begin{array}{l} \text{Helvetica} \\ \text{other_font_name} \end{array} \right\}$	The font used for PostScript text output.
PrintDensity#: <i>integer</i>	$\left\{ \begin{array}{l} 50 \\ 100 \\ 150 \\ 300 \end{array} \right\}$	The plot resolution for PCL output.
PrintLineWidth#: <i>integer</i>	$\left\{ \begin{array}{l} (4 * \text{PrintDensity}) / 300 \\ \text{other_integer} \end{array} \right\}$	Width, in rasters, of generated lines for PCL output .

In the above resources, the # should be replaced with the digits 1 through 10 to specify which user selection is being defined. Below are two example definitions.

```
nastplot.PrintLabel1: PCL printer
nastplot.PrintFormat1: PCL
nastplot.PrintCommand1: lpr -Php %s
nastplot.PrintOrient1: Landscape
```

```
nastplot.PrintLabel2: PostScript file
nastplot.PrintFormat2: PostScript
nastplot.PrintOrient2: Portrait
```

Additional resource items are provided to control the display of **nastplot**. These are described in the following table.

RESOURCE	DESCRIPTION
TextBackground: <i>color_name</i>	Color of the background in text areas.
TextForeground: <i>color_name</i>	Color of the actual text.
ButtonBoxBackground: <i>color_name</i>	Color of the background in the areas containing the command buttons.
ButtonBackground: <i>color_name</i>	Color of the background of the buttons.
ButtonForeground: <i>color_name</i>	Color of the text on the buttons.
PlotBackground: <i>color_name</i>	Color of the graphics area background.
LineStyle#: <i>color_name</i>	The color of the line styles selected in UAI/NASTRAN . Line styles 1 through 10 are available.
ButtonFont: <i>font_name</i>	The font used on the button text.
TextFont: <i>font_name</i>	The font used for all other text.

A sample resource file is provided with the delivery and may be found in:

```
rel_dir/nastran#/nastplot.ad
```

There are default values for most of these resources, and you should not have to change them unless you need to define a printer or wish to change the display appearance. In this case the sample resource file can be modified as desired and used. These resources must be made available to the X Window System in any of the normal methods. Typically, the resources can also be made available to all users on your system by copying the resource file into the system application defaults directory. On most systems the file name you create should be as follows:

```
/usr/lib/X11/app-defaults/Nastplot
```

The resources can also be made available by adding the data to the **.xdefaults** or **.Xresources** file in each user's home directory. If your users do not have a file by this name, one can be created by copying the sample file mentioned above to the home directories, and then modifying these files as desired with any text editor.

nastplot is invoked with the command:

```
nastplot [ file_name ]
```

8.2.5 Special Versions of the **nastplot** Program

On HP/Apollo and Sun workstations special versions of **nastplot** are delivered which operate under the normal window system found on those computers (Display Manager and SunView respectively).

8.2.6 Summary of Available Plotting Programs

The following plot utility software is provided with **UAI/NASTRAN**, depending on your computer system as shown in the table below.

COMPUTER	WINDOW SYSTEM	SOURCE FILE	EXECUTABLE FILE
ALL	NONE	<code>tekplot.f</code> ¹	
ALL	NONE		<code>nastplotps</code>
Cray X-MP, Y-MP			
DECsystem DECstation			
HP 9000/700	X Window System		<code>nastplot</code>
IBM RS/6000	X Window System		<code>nastplot</code>
SGI (All)	IRIS Workspace System		<code>nastplot</code>
SUN (All)	SunView	<code>nastplot.f</code>	<code>nastplot</code>

1. Requires PLOT-10 Subroutine Library

8.3 THE UAI/NASTRAN PLOTTER COORDINATE SYSTEM

As with all graphic systems, the **UAI/NASTRAN** plots are created in a virtual coordinate system. By default, plots are assumed to be square in shape, with a high resolution of 65536 x 65536 *pixels* that has been selected so that maximum accuracy will be maintained regardless of your actual graphic device display resolution. This is shown in Figure 1a. The end-users of **UAI/NASTRAN** may include an **ASPECT RATIO** command in their data stream because they are using a particular paper size for a plotter or a nonsquare window on a raster display. You must insure that your plotter post-processor program maintains the specified **ASPECT RATIO** when mapping the virtual coordinates to the physical coordinates of your device so that you do not distort the resulting plot. The interpretation of the aspect ratio is shown in Figure 1b. As can be seen in this figure, the aspect ratio is reflected by changing the size of the virtual coordinate system.

8.4 THE PLOTTER CHARACTER REPRESENTATION

All character information within plots may be specified by the end-user in one of two manners: as **HARDWARE** characters, or as **STROKED** characters. If the end-user has selected the **STROKED** option, then **UAI/NASTRAN** creates the actual characters by drawing line segments. In this case, your post-processor does not explicitly process characters. It simply draws lines. On the other hand, if the end-user selects **HARDWARE**, then special character plot commands are created. You must then decide how to display the characters on your device based on the available graphics utility library.

If the **HARDWARE** option is used, each character is assumed to be in a *character box* whose default height, *char_ht*, is 1536 pixels, and whose default width, *char_wid*, is 768 pixels. The character box includes the actual character and vertical and horizontal spacing, *v_space* and

Figure 1. THE UAI/NASTRAN VIRTUAL COORDINATE SYSTEM

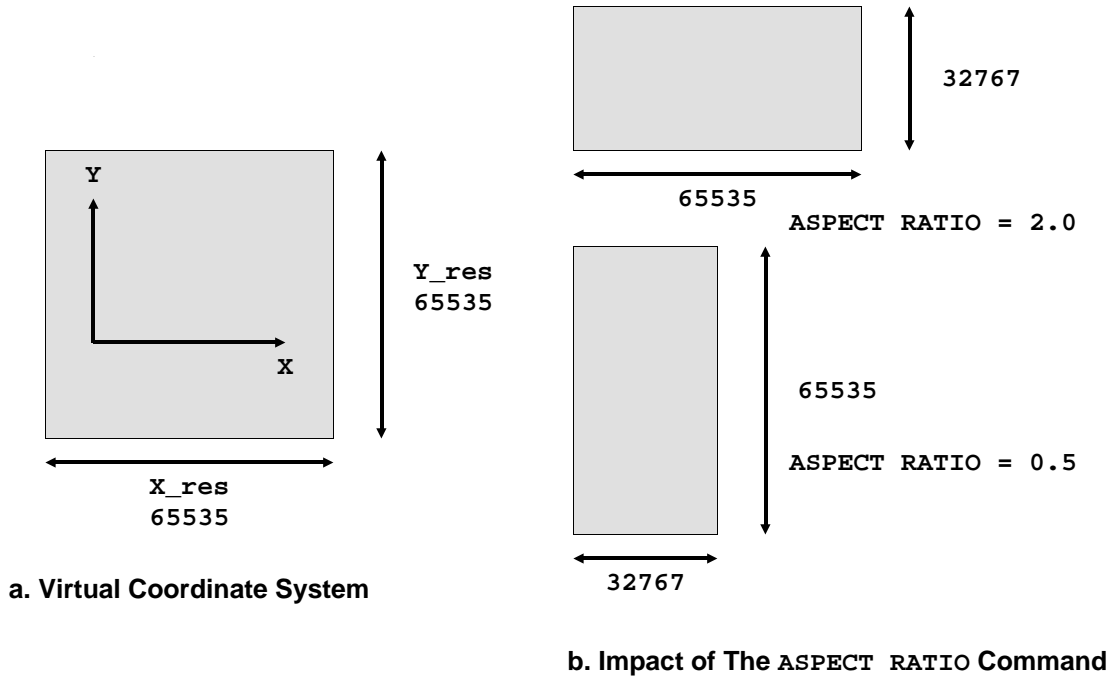
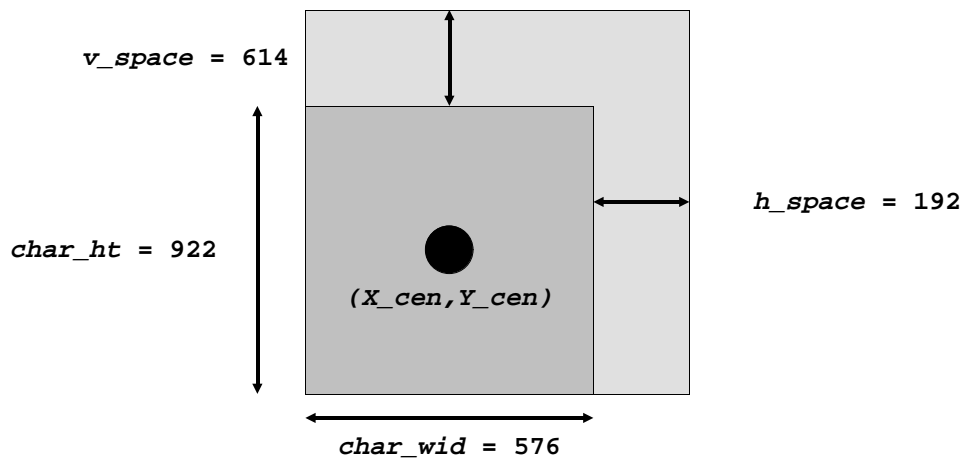


Figure 2. CHARACTER REPRESENTATION



h_space, respectively. This is shown in Figure 2. The end-user may change this size by using the **CHARACTER SCALE** command. The character is described by a character code and by the location of the center of the display area as shown in Figure 2.

It is crucial that you remember that the (X_{cen}, Y_{cen}) coordinates represent the middle of the character box as shown in Figure 2. Since different plotter utility libraries may base character positions from other points, such as lower left or bottom center, it may be necessary for you to transform the (X_{cen}, Y_{cen}) coordinates to these other values.

8.5 THE PLOTTER COMMANDS

The plot file contains a sequence of commands that define the graphic operations to be performed. Each command may be viewed as a record which has eight fields:

<i>plot_op</i>	<i>c_parm</i>	<i>R</i>	<i>S</i>	<i>T</i>	<i>U</i>	<i>V</i>	<i>W</i>
----------------	---------------	----------	----------	----------	----------	----------	----------

where *plot_op* is an integer representing a plot operation code, *c_parm* is an integer control parameter, and *R*, *S*, *T*, *U*, *V* and *W* are integer values whose meaning depends on the *plot_op*. Section 3.6 describes the two file formats that may be selected for the plot file.

There are only three primitive graphic operations that are used by the **UAI/NASTRAN** plotter:

<i>plot_op</i>	NAME	DESCRIPTION
0 or 2 or 3	<i>no_op</i>	Indicate null operations.
1	<i>start_plot</i>	Starts a new plot and provides resolution information.
4 or 14	<i>char</i>	Defines a character and its location.
5 or 15 6 or 16	<i>draw_line</i>	Defines a line and its beginning and end points.

8.5.1 The *start_plot* Plot Operation

The *start_plot* command indicates the beginning of a new plot. The general form of the command is:

1		<i>plot_num</i>	<i>X_res</i>	<i>Y_res</i>		<i>char_wid</i>	<i>char_ht</i>
---	--	-----------------	--------------	--------------	--	-----------------	----------------

where the *plot_num* is an integer value that gives the plot identification number. The pair of integers (X_{res}, Y_{res}) indicate the virtual resolution of the plot. Although the default resolution is 65536 by 65536, the end-user may select a nonsquare aspect ratio. In this case you must be careful to insure that your plot program does not distort the aspect ratio expected by the end-user. Finally, the pair of integer values $(char_wid, char_ht)$ define the character box size as described in Section 3.4.

8.5.2 The *char* Plot Operation

The *char* command indicates that your program must place a character or symbol at a particular location on the plot. Most often, there are large blocks of *char* commands. The first command in the block has the special *plot_code* 14 while those following have the *plot_code* of 4. The general form of the *char* commands is:

14	<i>c_code_1</i>	<i>X_cen_1</i>	<i>Y_cen_1</i>	<i>pos_1</i>	<i>len_str</i>		
4	<i>c_code_2</i>	<i>X_cen_2</i>	<i>Y_cen_2</i>	<i>pos_2</i>	<i>len_str</i>		
4	<i>c_code_3</i>	<i>X_cen_3</i>	<i>Y_cen_3</i>	<i>pos_3</i>	<i>len_str</i>		
4	<i>continues with char commands</i>						

where *c_code* is an integer value that is a character code specifying the symbol to be plotted. The available character codes are shown in Table 1. The pair of values (*X_cen*, *Y_cen*) defines the virtual coordinates of the center of the character box as defined in Section 3.4. The next two fields are used when the string of characters is to be plotted horizontally on the plotting surface. In such cases, *pos* specifies the position in the string of symbols whose length is *len_str*. For characters that will be plotted vertically, the values of *pos* and *len_str* are always 1. This allows your post-processing program to accumulate a single string of characters to be plotted in a single operation. In many cases, your plotter utility library will generate more readable text when you use this procedure.

Table 1. PLOTTING CHARACTER CODES

<i>c_code</i>	CHARACTER	<i>c_code</i>	CHARACTER	<i>c_code</i>	CHARACTER	<i>c_code</i>	CHARACTER
1	0	14	D	27	Q	40	-
2	1	15	E	28	R	41	*
3	2	16	F	29	S	42	/
4	3	17	G	30	T	43	=
5	4	18	H	31	U	44	.
6	5	19	I	32	V	45	,
7	6	20	J	33	W	46	\$
8	7	21	K	34	X	47	'
9	8	22	L	35	Y	48	.
10	9	23	M	36	z	49	O
11	A	24	N	37	(50	□
12	B	25	O	38)	51	◇
13	C	26	P	39	+	52	△

8.5.3 The *draw_line* Plot Operation.

The *draw_line* operation combines the concepts of moving and drawing. Most often, there are large blocks of *draw_line* commands. The first command in the block has the special *plot_code* 15 while those following have the *plot_code* of 5. The general form of the *draw_line* commands is:

15	<i>l_style</i>	<i>X_start_1</i>	<i>Y_start_1</i>	<i>X_end_1</i>	<i>Y_end_1</i>		
5	<i>l_style</i>	<i>X_start_2</i>	<i>Y_start_2</i>	<i>X_end_2</i>	<i>Y_end_2</i>		
5	<i>l_style</i>	<i>X_start_3</i>	<i>Y_start_3</i>	<i>X_end_3</i>	<i>Y_end_3</i>		
5	<i>continues with draw_line commands</i>						

where the *l_style* is an integer value that was selected by the user with the **LINE STYLE** command. You must determine the types of line styles that you wish your plot program to support and provide these to the users. Depending on your target display device, *l_styles* could include various thicknesses, dash patterns, colors, or any other attribute you wish. The pair of values (*X_start*, *Y_start*) defines the virtual coordinates of the starting point of the line, while (*X_end*, *Y_end*) define the ending coordinates. Naturally, to perform this operation in your plot program, you most often must move to the starting point and then draw to the ending point.

8.6 THE PLOT FILE FORMAT

There are two **UAI/NASTRAN** plot file formats: binary and formatted. The format is selected in the Executive Control command packet at the time the plot file is **ASSIGNED**. For plot files, the general format of the **ASSIGN** command is:

```
ASSIGN logical_name [ = phys_name ] ,NEW,USE=PLOT [ ,REALLOC ]
      [ ,TYPE = { BINARY
                  FORMATTED } ]
```

The key attribute to consider is the **TYPE**. Either **FORMATTED** or **BINARY** may be selected; and **BINARY** is the default. The format of each of these files is described in the next two sections.

8.6.1 The FORMATTED Plot File

The **FORMATTED** plot file is written using the Fortran format statement:

```
FORMAT( 2I3 , 4I6 , 2I5 )
```

This creates a file which contains one plot command for each record. The eight fields correspond to the fields shown in Section 3.5. As you can see from the sample data in Figure 3, you may readily examine the file in your text editor. To use the file, you simply read the data into variables which represent the fields of each record and then use them. The input of data is terminated when you encounter an end-of-file condition.

8.6.2 The BINARY Plot File

The **BINARY** plot file is available to support plotting post-processors that were developed prior to **UAI/NASTRAN** Version 11.0, or for programs written to support other **NASTRAN** variants. The **BINARY** file contains records which are 3000 bytes long. Each plotter command is stored in 30 bytes. The *plot_op* and *c_parm* plotter command fields are stored as integers one byte long. These fields may be extracted directly. Unlike the **FORMATTED** file, the remaining data fields in the **BINARY** file is represented in binary coded decimal. For example, the **R** field in the plot command is represented by 5 bytes as:

$$R_4R_3R_2R_1R_0$$

To determine the numeric value of these 5 bytes, you must compute:

$$R = R_410^4 + R_310^3 + R_210^2 + R_110^1 + R_010^0$$

The *S*, *T*, *U*, *V* and *W* fields must be decoded in a similar fashion. However, *V* and *W* are each represented by 4 digits rather than 5.

The **tekplot** source program provides code examples for processing these data. Also, if you are writing a plotter post-processor for the first time, you are encouraged to use the **FORMATTED** plot file to greatly simplify your coding and testing.

Figure 3. EXAMPLE CHARACTER PLOT FILE DATA

```

1  0 000001 65536 65536 00000 0576 0921
2  2 00000 00000 00000 00000 0000 0000
3  2 00000 00000 00000 00000 0000 0000
2  2 00000 00000 00000 00000 0000 0000
14 2 00768 64000 00001 00001 0000 0000
4  2 07680 64000 00002 00008 0000 0000
4 42 08448 64000 00003 00008 0000 0000
4  3 09216 64000 00004 00008 0000 0000
4  4 09984 64000 00005 00008 0000 0000
4 42 10752 64000 00006 00008 0000 0000
4 10 11520 64000 00007 00008 0000 0000
4  3 12288 64000 00008 00008 0000 0000
4  2 64768 64000 00001 00001 0000 0000
14 32 01536 06144 00001 00048 0000 0000
4 19 02304 06144 00002 00048 0000 0000

```

APPENDIX A.

THE CONFIGURATION SECTIONS

This Appendix presents a detailed description of the Preference File parameters for each of the UAI software products.

A.1 NOMENCLATURE

The table below presents the symbols used in describing the Preference File parameters in the remainder of this Appendix.

SYMBOL	DESCRIPTION
<i>char(len)</i>	Indicates that the parameter is a character string of maximum length <i>len</i> . Character strings need to be enclosed in single quotation marks only if they contain embedded blanks or special characters: <code>MODEL='IBM RS/6000'</code>
<i>int_val</i>	Indicates that the parameter is an integer value.
<i>real_val</i>	Indicates that the parameter is a real value. These values must be entered in decimal format, i.e. <code>1000000.0</code> , or in exponential notation such as <code>1.0E+7</code> .
<i>path_loc</i>	Indicates that the parameter is a Unix path name which must be enclosed in single quotation marks: <code>Perm-eBASE-Data-Loc='/big/disk/perm/storage'</code>
<i>template</i>	Indicates that the parameter is a file name template. The asterisk in the template is the placeholder for the substitution.
<i>db_name</i>	Indicates that the parameter is a physical name for a database including any path information.

SYMBOL	DESCRIPTION
<i>file_name</i>	Indicates that the parameter is a physical name for a file.
<i>mem_val</i>	Indicates a memory value which can take one of the following forms xM words * 1000000 xMW words * 1000000 xMB bytes * 1000000 xK words * 1000 xKW words * 1000 xKB bytes * 1000 x words xW words xB bytes

A.2 THE HOST CONFIGURATION SECTION

This section includes parameters which identify the type of computer, license information, and override Preference File information.

A.2.1 Site Description

The following parameters describe the host computer for the UAI Software Products.

- `Manufacturer = char(24)`

Specifies the manufacturer of your host computer.

- `Model=char(24)`

Specifies the model of your host computer.

- `Site = char(24)`

Identifies your site. You may wish to change this if you prefer one different from that selected by UAI at delivery time, or if you are installing multiple systems delivered under a site license.

- `License = char(24)`

Specifies your UAI license number. This parameter should not be changed.

A.2.2 Preference Override Information

- `Sys-Pref-File-Loc = 'template'`
- `User-Pref-File-Loc = 'template'`
- `Local-Pref-File-Loc = 'template'`

These parameters specify templates that are used to build the full path name for the System, User, and Local Preference Files which are used to

override the default configuration. Each of these parameters must contain an asterisk (*) which will be replaced by a character string to generate the actual file name. This string is provided by the specific program. The exact method each application uses to provide these substitution strings is described in subsequent Chapters of this manual.

- `Sys-Pref-Default-Name = char`
- `User-Pref-Default-Name = char`
- `Local-Pref-Default-Name = char`

These parameters provide the default substitution strings that are used to create the file names for the System, User, and Local Preference Files using the templates described in the previous section. These are only used if the application does not provide its own substitution string. If any of these parameters are not defined, and the application does not provide a substitution string, then **eBase** will not search for an override preference file of that type.

The System Preference File may itself override the parameters used to generate the User and Local Preference File names. Similarly, the User Preference File may override parameters in the Local Preference File.

A.3 THE *eBase* CONFIGURATION SECTION

This section describes the parameters in the **eBase** Configuration Section. Specific applications using the **eBase** kernel, including **UAI/NASTRAN**, **ASTROS**, and **eShell**, allow you to change the value of some of these parameters. The methods to do this are described in later Chapters.

A.3.1 Computing Resources

The following parameters control the use of system resources by the **eBase** kernel.

- `eBase-Initial-Memory = mem_val`
- `eBase-Memory-Increment = mem_val`
- `eBase-Max-Memory = mem_val`

These three parameters control the amount of memory that the **eBase** kernel may use for database operations while executing. The first is `eBase-Initial-Memory`. This defines the initial amount of memory that the database kernel will use. The second is `eBase-Memory-Increment` which provides an optional memory increment size. If memory is exhausted during execution, and an increment size has been provided, then the memory will be extended by the specified value. This procedure will continue as necessary until `eBase-Max-Memory` is reached. For example:

```
eBase-Initial-Memory = 100kw
eBase-Memory-Increment = 50kw
eBase-Max-Memory = 5mw
```

initially allocates 100,000 single precision words of memory to **eBase**. If additional memory is required, it is added in 50,000 word increments until the maximum of 5 million words is obtained. The amount of **eBase** memory required depends on many factors including: the Index and Data Component File block sizes, the number of databases simultaneously attached, and the number of entities open at a given time. The actual units available to define the memory sizes are described in Chapter 2.

These two configuration parameters control your system defaults for database block sizes. These are:

- `Index-File-BlockSize = int_val`
`Data-File-BlockSize = int_val`

See the section earlier in this Chapter that discusses the proper selection of block sizes. You may determine a value by testing several examples of production databases being run on your computer. All block size values are specified in single precision words. Note that the block size parameters may be rounded up for improved I/O efficiency on some host computers.

- `Perm-eBASE-Integrity = { High }
 { Low }`
- `Temp-eBase-Integrity = { High }
 { Low }`

These two parameters are used to determine the amount of integrity that is maintained for **eBase** databases by selecting the frequency of I/O operations used to synchronize the physical database files with the contents of memory buffers. If you are providing your own integrity through stringent backup procedures or by routinely writing database files to tapes, then you can improve performance by selecting the `Low` option. For better integrity, you select `High`. The `Temp-eBase-Integrity` parameter is provided for completeness; it should be set to `Low` in all circumstances presently anticipated. Specifies the integrity level for temporary **eBase** databases.

- `Max-Index-Building-Memory = mem_val`

Specifies the maximum amount of memory that **eBase** will use when creating indices on relational entities.

- `Optimize-Index-Building = { Time }
 { Memory }
 { Both }`

Specifies whether **eBase** should use less memory to build indexes, which takes more **TIME**, or if it should use more **MEMORY**, which takes less time. This parameter only impacts the creation of indexes. The access performance of the resulting relations is the same regardless of the method selected.

A.3.2 I/O System Parameters

Parameters which control the location of the various database files are described below:

➤ Perm-eBase-Index-Loc = '*path_loc*'

Specifies the location where permanent database Index Component files will be placed.

➤ Perm-eBase-Data-Loc = '*path_loc1*'

Perm-eBase-Data-Loc = '*path_loc2*'

...

Specifies the names of one or more locations where permanent database Data Component files will reside. Multiple copies of this line are used to define multiple directory names.

➤ Temp-eBase-Index-Loc = '*path_loc*'

Specifies the name of the location where temporary database Index Component files will reside.

➤ Temp-eBase-Data-Loc = '*path_loc_1*'

Temp-eBase-Data-Loc = '*path_loc_2*'

...

Specifies the names of one or more locations where temporary database Data Component files will reside. Multiple copies of this line are used to define multiple directory names.

In order to improve **eBase** performance and make optimal use of available disk resources, you may direct the Index Component File and the Data Component Files to specific file systems. There are separate controls for permanent databases and for temporary databases. These configuration parameters are:

Perm-eBase-Index-Loc = '*path_loc*'

Perm-eBase-Data-Loc = '*path_loc*'

Temp-eBase-Index-Loc = '*path_loc*'

Temp-eBase-Data-Loc = '*path_loc*'

For these configuration parameters, you specify a complete *path_loc* name which specifies the directory in which the indicated file, or files, will reside. In the case of the Data File Components, both for permanent and temporary databases, you may specify a list of one or more locations. This is done by listing them sequentially:

Perm-eBase-Data-Loc = '*path_loc1*'

Perm-eBase-Data-Loc = '*path_loc2*'

Perm-eBase-Data-Loc = '*path_loc3*'

- File-Utilization = $\left\{ \begin{array}{l} \text{Fill-First} \\ \text{Round-Robin} \end{array} \right\}$

The `Fill-First` option indicates that the Data File Components will be used in the specified order one after the other as they fill their allocated file systems. The `Round-Robin` option requests that all files be used circuitously, sometimes called *striping*.

A.3.3 Program Authorization

The following parameters are used for **eBase** kernel authorization and should only be modified under instruction by UAI.

- EBSPAK01 = 'keydata'
- EBSPAK02 = 'keydata'
- ...

Defines the Program Access Key data that is used to validate your use of the **eBase** kernel.



Do not change these data unless instructed by UAI. If they are changed, all applications using the eBase kernel will not function.

A.4 THE UAI/NASTRAN CONFIGURATION SECTION

This section describes the **UAI/NASTRAN** Configuration Section. These parameters define default values for system and engineering data which are unique to **UAI/NASTRAN**.

A.4.1 Print File Controls

The following parameters control the appearance and contents of the print file generated by **UAI/NASTRAN**.

- Lines-per-Page = *int_val*

Specifies the number of lines per page that will be printed. Typically, you must change this value depending on the size of your paper, the number of lines printed per inch, or whether your printer is a line printer or laser printer.

- Maximum-Print-Lines = *int_val*

Specifies the default value for the maximum number of lines of print for users who do not use the Executive Control command `MAXLINES`.

- Diag-Output = $\left\{ \begin{array}{l} \text{Log} \\ \text{Print} \end{array} \right\}$

Specifies where **DIAG 8** and **DIAG 19** messages are written. The default value writes them in the **PRINT** file. If your users want compatibility with other NASTRAN variants, you may select the **LOG** file.

- `Default-Diag = (int_list)`

Specifies default values for the Executive Control **DIAG** command.

- $$\text{BulkData-Echo} = \left\{ \left\{ \begin{array}{l} \text{Both} \\ \text{None} \\ \text{Sort} \\ \text{Unsort} \end{array} \right\}, \left\{ \begin{array}{l} \text{Punch} \\ \text{Nopunch} \end{array} \right\} \right\}$$

Provides default values for the Case Control command **ECHO**.

- `News-File = file_name`

File which contains **UAI/NASTRAN** news to be printed at the start of each run.

A.4.2 Computing Resources

The following parameters control the use of system resources.

- `CPU-Time = int_val`

Specifies the default CPU time, in minutes, given to users who do not use the Executive Control command **TIME**.

- `Working-Memory = mem_val`
- `Maximum-Memory = mem_val`
- `Physical-Memory = mem_val`

These three parameters control the amount of working memory used by **UAI/NASTRAN**. The `Working-Memory` parameter specifies the default. It may be overridden by the `-m` option of the `uainast` script or the Executive Control Command **MEMORY**. The `Maximum-Memory` parameter specifies the largest value of working memory that a user is allowed to request. The `Physical-Memory` parameter is used to control the use of system resource and was discussed in detail earlier in this chapter. Its value may also be overridden with the Executive Control Command **MEMORY**. The actual units available to define the memory sizes are described in Chapter 2.

- `Dynamic-Initial-Memory = mem_val`
- `Dynamic_Memory-Increment = mem_val`
- `Dynamic-Max-Memory = mem_val`

Several portions of **UAI/NASTRAN** use small amounts of **eBase:applib** memory that is separate from the large working memory, or the memory that the **eBase** kernel uses. These parameters serve the same purpose as those of the same name in the **eBase:applib** configuration section. See Chapter 5 of this manual for a description of these parameters.

A.4.3 Matrix Conditioning

The following parameters are used to control some of the matrix conditioning features of **UAI/NASTRAN**. These features deal with the finite element model. Other purely mathematical conditioning of matrices are controlled in the **matlib** Configuration Section, see Chapter 6.

➤ $\text{AutoSPC-Select} = \begin{Bmatrix} \text{Yes} \\ \text{No} \end{Bmatrix}$

Specifies the **AUTOSPC** selection default option, selected from **Yes** or **No**.

➤ $\text{AutoSPC-Method} = \begin{Bmatrix} \text{MPC} \\ \text{SPC} \end{Bmatrix}$

Specifies the **AUTOSPC** method default option. If your users are using other NASTRAN variants, you may change **MPC** to **SPC** for compatibility.

➤ $\text{AutoSPC-Print} = \begin{Bmatrix} \text{Yes} \\ \text{No} \end{Bmatrix}$

Selects the **AUTOSPC** print default option. If your users do not want this print option, you may replace **Yes** with **No**.

➤ $\text{AutoSPC-Eps} = \text{real_val}$

Specifies the **AUTOSPC** default stiffness ratio value.

➤ $\text{NonLinear-Nset-Autospc} = \begin{Bmatrix} \text{Yes} \\ \text{No} \end{Bmatrix}$

Selects, or deselects, **AUTOSPC** processing on the *n-set* for nonlinear analyses.

➤ $\text{Sequencer-Method} = \begin{Bmatrix} \text{Best} \\ \text{CM} \\ \text{GPS} \\ \text{MWF} \\ \text{RQT} \\ \text{SND} \\ \text{None} \end{Bmatrix}$

Selects a default method value for the Grid Point Sequencer module. This value should not be changed without consultation with UAI.

- $\text{Sequencer-Criteria} = \left\{ \begin{array}{l} \text{Bandwidth} \\ \text{Maxwave} \\ \text{Profile} \\ \text{RMSWave} \end{array} \right\}$

Selects a default criteria value for the Grid Point Sequencer module. This value should not be changed without consultation with UAI.

- $\text{Sequencer-Print} = \left\{ \begin{array}{l} \text{Detail} \\ \text{None} \\ \text{Summary} \end{array} \right\}$

Selects a default print option for the Grid Point Sequencer module. This setting may be overridden by use of the Executive Control command **SEQUENCER**.

For a more detailed description of these values, see the **UAI/NASTRAN User's Reference Manual** sections on the **AUTOSPC** Case Control Command and the **SEQUENCER** Executive Control Command.

A.4.4 Data Checking

UAI/NASTRAN performs extensive data checking of the user's input data stream. The following parameters are used to control this checking.

- $\text{Solid-Geom-Check} = \left\{ \begin{array}{l} \text{Fatal} \\ \text{Warning} \\ \text{Ignore} \end{array} \right\}$
- $\text{Solid-Geom-Percentage} = \text{real_val}$
- $\text{Solid-Geom-MaxWarnings} = \text{int_val}$

Provide default values for the Executive Control command: **DATACHECK SOLIDGEOM**.

- $\text{Solid-Geom-Max-Percentage} = \text{real_val}$

Defines the maximum value allowed for the **Solid-Geom-Percentage** parameter.

- $\text{Plate-Geom-Check} = \left\{ \begin{array}{l} \text{Fatal} \\ \text{Warning} \\ \text{Ignore} \end{array} \right\}$
- $\text{Plate-Geom-Percentage} = \text{real_val}$
- $\text{Plate-Geom-MaxWarnings} = \text{int_val}$

Provides default values for the Executive Control command: **DATACHECK PLATEGEOM**.

- $\text{Plate-Geom-Max-Percentage} = \text{real_val}$

Defines the maximum value allowed for the `Plate-Geom` parameter.

➤ `Beam-Offset-Check` = $\left\{ \begin{array}{l} \text{Fatal} \\ \text{Warning} \\ \text{Ignore} \end{array} \right\}$

➤ `Beam-Offset-Ratio` = *real_val*

➤ `Beam-Offset-MaxWarnings` = *int_val*

Provides default values for the Executive Control command: **DATACHECK BEAMOFFSETS.**

➤ `Plate-Offset-Check` = $\left\{ \begin{array}{l} \text{Fatal} \\ \text{Warning} \\ \text{Ignore} \end{array} \right\}$

➤ `Plate-Offset-Ratio` = *real_val*

➤ `Plate-Offset-MaxWarnings` = *int_val*

Provides default values for the Executive Control command: **DATACHECK PLATEOFFSETS.**

➤ `Plate-Aspect-Ratio-Check` = $\left\{ \begin{array}{l} \text{Fatal} \\ \text{Warning} \\ \text{Ignore} \end{array} \right\}$

➤ `Plate-Aspect-Ratio-Ratio` = *real_val*

➤ `Plate-Aspect-Ratio-MaxWarnings` = *int_val*

Provides default values for the Executive Control command: **DATACHECK ASPECTRATIO.**

➤ `Quad-Warping-Check` = $\left\{ \begin{array}{l} \text{Fatal} \\ \text{Warning} \\ \text{Ignore} \end{array} \right\}$

➤ `Quad-Warping-Ratio` = *real_val*

➤ `Quad-Warping-MaxWarnings` = *int_val*

Provides default values for the Executive Control command: **DATACHECK QUADWARP.**

➤ `Constraint-Errors` = $\left\{ \begin{array}{l} \text{Fatal} \\ \text{Warning} \\ \text{Ignore} \end{array} \right\}$

➤ `Constraint-MaxWarnings` = *int_val* Provides default values for the Executive Control command: **DATACHECK CONSTRAINTS.**

Provides default values for the Executive Control command: **DATACHECK CONSTRAINTS.**

- `Coordinate-Errors` = $\left\{ \begin{array}{l} \text{Fatal} \\ \text{Warning} \\ \text{Ignore} \end{array} \right\}$
- `Coordinate-Errors-Treatment` = $\left\{ \begin{array}{l} \text{Basic} \\ \text{Rectangular} \end{array} \right\}$

A more detailed description of most of these values can be found in the **UAI/NASTRAN** User's Reference Manual section on the **DATACHECK** Executive Control Command.

A.4.5 Solution Techniques

The following parameters are used to control some aspects of the **UAI/NASTRAN** solution techniques.

- `Eigen-Normalization` = $\left\{ \begin{array}{l} \text{Mass} \\ \text{Max} \end{array} \right\}$

Provides a default Eigenvalue extraction normalization rule. This may be overridden by the user with the **EIGR** Bulk Data entry.

- `Mass-Orthogonality-Test` = *real_val*

Defines the default value for the mass orthogonality test for the Lanczos and Givens eigenextraction methods. The value delivered is 0.0 which means that no checks are made. This saves computation and allows compatibility with MSC/NASTRAN.

- `Default-Grid-Temperature` = $\left\{ \begin{array}{l} \text{None} \\ \text{real_val} \end{array} \right\}$

Specifies the default temperature at Grid points which have no temperature defined. If this parameter has a *real_val*, then the **TEMPD** Bulk Data entry is not required. If **NONE** is selected, then **TEMPD** data are required.

A.4.6 Element Options

The following parameters are used to control the formulation used by some of the finite elements in **UAI/NASTRAN**.

- `K6Rot` = *real_val*

Provides a default value for Bulk Data entry **PARAM, K6ROT**

- `Hexa-Bubble` = $\left\{ \begin{array}{l} \text{3x6} \\ \text{6x21} \end{array} \right\}$

Defines the type of bubble functions used by the **HEXA** element formulation.

- `Beam-Automatic-Warping` = $\left\{ \begin{array}{l} \text{No} \\ \text{Yes} \end{array} \right\}$

Defines how BEAM elements with nonzero warping coefficients, **CWA** or **CWB**, but without warping degrees of freedom, **WIDA** and **WIDB**, are treated. A value of **YES** will cause the warping DOF to be automatically generated, a value of **NO** will cause the warping to be ignored.

$$\text{➤ Linear-Plate-Center-Stress} = \left\{ \begin{array}{c} \text{Best} \\ \text{Average} \\ \text{Direct} \end{array} \right\}$$

Defines the method used to compute center stresses for plate elements. A value of **AVERAGE** uses the average of the stresses at the integration points; **DIRECT** results in the computation of the stress directly at the center of the element; and **BEST** selects the best method for each element.

$$\text{➤ Linear-Plate-Corner-Stress} = \left\{ \begin{array}{c} \text{Required} \\ \text{Always} \end{array} \right\}$$

Control wheter corner stresses are computed for the TRIA3, TRIAR, QUAD4 and QUADR elements. A value of **ALWAYS** requests that they always be computed. A value of **REQUIRED** indicates that they will be computed only if they are required for subsequent computations, such as GRID Point Stresses.

$$\text{➤ Quad-Warping-Threshold} = \text{real_val}$$

Sets the threshold for which warping will be corrected for QUAD4 and QUADR elements.

A.4.7 Analysis Output Control

The following parameters is used to control some aspects of the type of solution output from **UAI/NASTRAN**.

$$\text{➤ Plate-Stress-Coordinate} = \left\{ \begin{array}{c} \text{Element} \\ \text{Material} \\ \text{Basic} \\ \text{Post} \end{array} \right\}$$

Provides a default type of coordinate system used for stress output of **PLATE** elements.

$$\text{➤ Solid-Material-Coordinate} = \left\{ \begin{array}{c} \text{Element} \\ \text{Basic} \end{array} \right\}$$

Provides a default type of material coordinate system used for **SOLID** elements.

$$\text{➤ Solid-Stress-Coordinate} = \left\{ \begin{array}{c} \text{Element} \\ \text{Material} \\ \text{Basic} \end{array} \right\}$$

Provides a default type of coordinate system used for stress output of **SOLID** elements.

- `Element-KE-Threshold = real_val`

Provides a default value for the **THRESH** option of Case Control command **EKE**.

- `Element-SE-Threshold = real_val`

Provides a default value for the **THRESH** option of Case Control command **ESE**.

A.4.8 I/O System Parameters

UAI/NASTRAN allows you to control the location and block size characteristics of interface files, the location of the system database, and the location of the **DMAP ALTER** library. These are summarized in the following table.

- `AlterLib-Loc = template`

Specifies the directory in which you place the **ALTERLIB**. Note that this parameter must contain a single asterisk (*) as it appears in the `uaidef` file.

- `External-File-BlockSize = int_val`

Specifies the block size, in single precision words, used for writing a number of **UAI/NASTRAN** interface files.

- `External-Temp-Loc = path_loc`

Specifies the name of the directory where temporary interface files will be placed during a **UAI/NASTRAN** execution.

- `System-Database-Loc = path_loc`

Specifies the name of the directory where the **UAI/NASTRAN** system database resides.

- `Maximum-Filename-Length = int_val`

Specifies the maximum length allowed for each file name component of the `phys_name` parameter of the **ASSIGN** Command.

A.4.9 Assign Processing

The following parameters are used to control the assignment of files in **UAI/NASTRAN**.

- `Auto-Assign = template`

Defines the default **ASSIGNS**. to be used by **UAI/NASTRAN**.

- Logical-Unit-Outputx = *char(8)*

Provides the default logical unit names for the **OUTPUTx** modules.

- Logical-Unit-Inputtx = *char(8)*

Provides the default logical unit names for the **INPUTTx** modules.

- Upper-Case-Assign = $\left\{ \begin{array}{l} \text{Yes} \\ \text{No} \end{array} \right\}$

Controls whether any Executive Control **ASSIGN** and **INCLUDE** commands are converted to upper case.

A.4.10 Index Archive Control

The following parameters are used to control database indexing options.

- Auto-Index-Archive = $\left\{ \begin{array}{l} \text{Yes} \\ \text{No} \end{array} \right\}$

- Index-Archive = $\left\{ \begin{array}{l} \text{Yes} \\ \text{No} \end{array} \right\}$

- Unique-Index-Archive-Attribute = $\left\{ \begin{array}{l} \text{'Entity_name = att1,att2,...'} \\ \text{'Entity_name = (att1,att2,..)'} \\ \text{'att1,att2,...'} \end{array} \right\}$

- Nonunique-Index-Archive-Attribute = $\left\{ \begin{array}{l} \text{'Entity_name = att1,att2,...'} \\ \text{'Entity_name = (att1,att2,..)'} \\ \text{'att1,att2,...'} \end{array} \right\}$

These parameters are used to control the generation of indices for relational entities on the **ARCHIVE** database. *Entity_name* indicates the name of a relational entity on the database, and *att1,att2* the names of attributes in a relational schema. **UAI/NASTRAN** has a default set of indices for all of the relations. This is controlled with the parameter **Auto-Index-Archive**. To disable this feature, you select the **NO** option for this parameter. You may then manually specify the indices to be built by enabling the **Index-Archive** command. This will allow you to use the next two parameters.

The parameters **Unique-Index-Archive-Attribute** and **Nonunique-Index-Archive-Attribute** are used to specify explicit attributes that will be indexed. There are three options based on the form of the parameter:

- The form *'Entity_name=att1,att2,...'* results in a **separate index** being built for each listed attribute in the specified relation.
- The form *'Entity_name=(att1,att2,...)'*, where the attribute names are enclosed in parentheses, results in a **multiple attribute index** being built for the set of attributes in the specified relation.
- The form, *'att1,att2,...'* results in an index being built for every entity which has one of the listed attributes.

A.4.11 Program Authorization

The following parameters are used for **UAI/NASTRAN** program authorization and should only be modified under instruction by UAI.

- `NASPAK01 = 'keydata'`
- `NASPAK02 = 'keydata'`
- ...

Defines the Program Access Key data that is used to validate your use of **UAI/NASTRAN**.



Do not change these data unless instructed by UAI. If they are changed, the program will not function.

A.5 THE *ASTROS* CONFIGURATION SECTION

This section describes the ***ASTROS*** Configuration Section. These parameters define default values for system and engineering data which are unique to ***ASTROS***.

A.5.1 Print File Controls

The following parameters control the appearance and contents of the print file generated by ***ASTROS***.

- `Lines-per-Page = int_val`

Specifies the number of lines per page that will be printed. Typically, you must change this value depending on the size of your paper, the number of lines printed per inch, or whether your printer is a line printer or laser printer.

- `Maximum-Print-Lines = int_val`

Specifies the default value for the maximum number of lines of print.

- `BulkData-Echo = { { Echo }, { Sort }, { Print }
{ Noecho }, { Nosort }, { Punch }
{ Both } }`

Provides default values for the Begin Bulk echo options.

A.5.2 Computing Resources

The following parameters control the use of system resources.

- Working-Memory = *mem_val*
- Maximum-Memory = *mem_val*
- Physical-Memory = *mem_val*

These three parameters control the amount of working memory used by **ASTROS**. The `Working-Memory` parameter specifies the default. It may be overridden by the `-m` option of the `astros` script or the Executive Control Command `MEMORY`. The `Maximum-Memory` parameter specifies the largest value of working memory that a user is allowed to request. The `Physical-Memory` parameter is used to control the use of system resource and was discussed in detail earlier in this chapter. Its value may also be overridden with the Executive Control Command `MEMORY`. The actual units available to define the memory sizes are described in Chapter 2.

A.5.3 Matrix Conditioning

The following parameters are used to control some of the matrix conditioning features of **ASTROS**. These features deal with the finite element model. Other purely mathematical conditioning of matrices are controlled in the *matlib* Configuration Section, see Chapter 5.

- AutoSPC-Select = $\left\{ \begin{array}{c} \text{Yes} \\ \text{No} \end{array} \right\}$

Specifies the `AUTOSPC` selection default option, selected from `yes` or `no`.

- AutoSPC-Method = $\{ \text{SPC} \}$

Specifies the `AUTOSPC` method default option. Currently, only `SPC` is available in **ASTROS**, but other options may be implemented later.

- AutoSPC-Print = $\left\{ \begin{array}{c} \text{Yes} \\ \text{No} \end{array} \right\}$

Selects the `AUTOSPC` print default option. If your users do not want this print option, you may replace `yes` with `no`.

- AutoSPC-Eps = *real_val*

Specifies the `AUTOSPC` default stiffness ratio value.

- Sequencer-Method = $\left\{ \begin{array}{c} \text{Best} \\ \text{CM} \\ \text{GPS} \\ \text{All} \\ \text{None} \end{array} \right\}$

Selects a default method value for the Grid Point Sequencer module. This value should not be changed without consultation with UAI.

$$\text{Sequencer-Criteria} = \left\{ \begin{array}{l} \text{Bandwidth} \\ \text{Maxwave} \\ \text{Profile} \\ \text{RMSWave} \end{array} \right\}$$

Selects a default criteria value for the Grid Point Sequencer module. This value should not be changed without consultation with UAI.

$$\text{Sequencer-Print} = \left\{ \begin{array}{l} \text{Detail} \\ \text{None} \\ \text{Summary} \\ \text{Diag} \end{array} \right\}$$

Selects a default print option for the Grid Point Sequencer module.

A.5.4 Data Checking

ASTROS performs extensive data checking of the user's input data stream. The following parameters are used to control this checking.

$$\text{Quad-Warping-Ratio} = \text{real_val}$$

Provides the value for the warping check for **QUAD4** elements. This number represents the distance between the two diagonals at their point where they cross on the element mean plane expressed as a fraction of the length of the longer diagonal.

$$\text{Undesigned-Stress-Constraints} = \left\{ \begin{array}{l} \text{Fatal} \\ \text{Warning} \\ \text{Ignore} \end{array} \right\}$$

Provides the value controlling whether stress constraints applied to undesigned elements should be ignored or considered an error. If an error, the error can be either a warning or a fatal error.

A.5.5 Solution Techniques

The following parameters are used to control some aspects of the **ASTROS** solution techniques.

$$\text{Eigen-Normalization} = \left\{ \begin{array}{l} \text{Mass} \\ \text{Max} \end{array} \right\}$$

Provides a default Eigenvalue extraction normalization rule. This may be overridden by the user with the **EIGR** Bulk Data entry.

$$\text{Panel-Buckling-Term-Select} = \left\{ \begin{array}{l} \text{Dynamic} \\ \text{Fixed} \end{array} \right\}$$

$$\text{Panel-Buckling-Initial-Terms} = \text{int_val}$$

- Panel-Buckling-Maximum-Terms = *int_val*
- Panel-Buckling-Dynamic-Target = *int_val*
- Panel-Buckling-Linearization-Power = *real_val*

Provides a selector for the method used in determining the number of terms to use in evaluating the buckling eigenvalue for buckling constraints defined on DCONBK Bulk Data entries. *Dynamic* means that the running load values and panel geometry will be used to select a number of series terms based on the values of *Maximum-Terms* and *Dynamic-Target*. *Fixed* means that an *n by n* set of terms will be used where *n* is taken from *Initial-Terms*. Finally, the *Linearization-Power* specifies the exponent used in linearizing the constraint function. These values should not be changed without consultation with UAI.

A.5.6 Element Options

The following parameters are used to control the formulation used by some of the finite elements in *ASTROS*.

- K6Rot = *real_val*

Provides a default value for Solution Control Command **K6ROT**=*real_val*

A.5.7 I/O System Parameters

ASTROS allows you to control the characteristics of interface files and the location of the system database. These are summarized in the following table.

- System-Database-Loc = *path_loc*

Specifies the name of the directory where the *ASTROS* system database resides.

- Maximum-Filename-Length = *int_val*

Specifies the maximum length allowed for each file name component of the *phys_name* parameter of the **ASSIGN** Command.

A.5.8 Optimization Control Options

The following parameters are used to establish default values for the Solution Control control **OPTIMIZE** command options. These commands are documented in the *ASTROS* User's Manual.

- Optimizer-Select = $\left\{ \begin{array}{l} \text{DOT} \\ \text{MDOT} \end{array} \right\}$

Selects the optimization kernel. MDOT is the default. The DOT option is available only for an additional cost and must be authorized by the Program Access Key (PAK).

- Optimization-Maxiter = *int_val*

Provides a default value for the OPTIMIZE MAXITER=*int_val* command option.

- MP-Move-Limit = *real_val*

Provides a default value for the OPTIMIZE MOVLIM=*real_val* command option.

- FSD-Move-Limit = *real_val*

Provides a default value for the OPTIMIZE ALPHA=*real_val* command option.

- Convergence-Limit = *real_val*

Provides a default value for the OPTIMIZE CONVRGLIM=*real_val* command option.

- Constraint-Retention-Factor = *real_val*

Provides a default value for the OPTIMIZE NRFAC=*real_val* command option.

- Constraint-Retention-Lbound = *real_val*

Provides a default value for the OPTIMIZE EPS=*real_val* command option.

- SA-Gradient-FDSTEPsize = *real_val*

Provides a default value for the OPTIMIZE FDSTEP=*int_val* command option.

- Max-Stationary-Obj = *int_val*

- Stationary-Deltaobj-value = *real_val*

Provides a value for the algorithm that determines if the objective function has become "frozen." If the change in the value of the objective is less than Stationary-Deltaobj-value for Max-Stationary-Obj iterations in a row, the program will terminate.

A.5.9 Program Authorization

The following parameters are used for *ASTROS* program authorization and should only be modified under instruction by UAI.

- `ASTPAK01 = 'keydata'`
- `ASTPAK02 = 'keydata'`
- ...

Defines the Program Access Key data that is used to validate your use of *ASTROS*.



Do not change these data unless instructed by UAI. If they are changed, the program will not function.

A.6 THE *eBase:applib* CONFIGURATION SECTION

The configuration parameters relating to the *eBase:applib* applications programming interface are discussed in this section. The following sections summarize the general parameters.

A.6.1 Computing Resources

- `Dynamic-Initial-Memory = mem_val`
- `Dynamic-Memory-Increment = mem_val`
- `Dynamic-Max-Memory = mem_val`

Specifies the maximum value that is available to the **DMMS**.

As with *eBase* kernel memory, there are three configuration parameters which control the allocation of Dynamic Memory from Unix to satisfy programmer requests, and their purpose parallels the corresponding *eBase* kernel parameters.

`Dynamic-Initial-Memory` is the minimum number of units which the Dynamic Memory Management System allocates from Unix to satisfy the first user request. `Dynamic-Memory-Increment` is the size of the additional pieces the **DMMS** allocates from Unix for requests which will not fit within the available pool. `Dynamic-Max-Memory` is an artificial limit which the **DMMS** will not exceed.

The actual units available to define the memory sizes are described in Chapter 2.

`Dynamic-Max-Memory` is artificial both because the hard limit is the memory available to the Unix process in which the user's program is requesting memory, and because the user may call `DMSIZE` to override all three of these configuration parameters.

The discussion below will use the terms **Initial Memory** (MEM_{init}), **Memory Increment** (MEM_{inc}), and **Max Memory** (MEM_{max}) to refer to the values in effect. Because the three values do not have to make sense together, the **DMMS** interprets them as follows. If:

$$\frac{MEM_{max} - MEM_{init}}{MEM_{inc}}$$

yields a fractional result, the fraction is ignored. In such a case, the effective upper limit is less than MEM_{max} because memory is only added to the MEM_{init} piece in whole MEM_{inc} pieces, and the **DMMS** will not allocate additional memory from Unix if the result would exceed MEM_{max} . The exact upper limit is:

$$\min \left[MEM_{max}, MEM_{init} + \text{int} \left[\frac{MEM_{max} - MEM_{init}}{MEM_{inc}} \right] MEM_{inc} \right]$$

where **int** indicates integer division.

A.7 THE *eBase:matlib* CONFIGURATION SECTION

The configuration parameters relating to the *eBase:matlib* utility library are discussed in this section. The following sections summarize these parameters.

A.7.1 Solver Options

- AutoSing-MaxCheck = $\left\{ \begin{array}{l} \text{Go} \\ \text{Nogo} \\ \text{None} \end{array} \right\}$
- AutoSing-MaxRatio = *real_val*

The AutoSing parameters tell the symmetric **LU** decomposition routine how to handle computed singularities. The manner in which this is done is discussed in the *eBase:matlib* Programmer's Manual.

- Solver-Select = $\left\{ \begin{array}{l} \text{Best} \\ \text{Standard} \\ \text{Sparse} \\ \text{MachDep} \end{array} \right\}$
- Sparse-Solver-Version = $\left\{ \begin{array}{l} \text{REL3} \\ \text{REL4} \end{array} \right\}$
- Solver-Print = $\left\{ \begin{array}{l} \text{None} \\ \text{Summary} \\ \text{Detail} \end{array} \right\}$
- Solver-Order = $\left\{ \begin{array}{l} \text{Sandard} \\ \text{Alternate} \\ \text{Best} \\ \text{None} \end{array} \right\}$

The `Solver-Select` parameter specifies which **eBase:matlib** linear equation solver to use. The available options include the high-performance sparse matrix solver (`Sparse`), the banded matrix solver (`Standard`), or the best of those available (`Best`). When selecting the `Sparse` option, you may also select a version of software from either Release 3 (`REL3`) or Release 4 (`REL4`). You should only change the delivered option on the advice of UAI Client Support.

Some vendors provide specialized solvers for their machines which may be supported by **eBase:matlib**. In such cases, the techniques are selected with the `MachDep` option. Note that the `Solver-Select` parameter may be overridden using the **eBase:matlib** subroutine `MLSCOPT`.

The `Solver-Print` parameter selects the amount of printed output the solver will produce. For the `Sparse` solver, the `Solver-Order` parameter selects which internal resequencing algorithm is used.

A.7.2 Timing Constants

- `Time-I/O = real_val`
- `Time-Matrix-Terms = (r1,r2)`
- `Time-Matrix-Columns = (r1,r2)`
- `Time-Matrix-Strings = (r1,r2)`
- `Time-Tight-Loop = (r1,r2,r3,r4)`
- `Time-Loose-Loop = (r1,r2,r3,r4)`

The timing parameters are used internally by **eBase:matlib** routines to determine specific algorithms for optimal performance. They are generated during the Installation Procedure and are specific to your host computer. They should not be modified without instructions from UAI.

A.7.3 Program Authorization

The following parameters are used for **eBase:matlib** program authorization and should only be modified under instruction by UAI.

- `MATPAK01 = 'keydata'`
- `MATPAK02 = 'keydata'`
- ...

Defines the Program Access Key data that is used to validate your use of **eBase:matlib** subroutines.



Do not change these data unless instructed by UAI. If they are changed, the program will not function.

A.8 THE *eShell* CONFIGURATION SECTION

This section describes the *eShell* Configuration parameters and indicates those which may be overridden by commands in *eShell*.

The configuration parameters relating to the *eShell* interactive interface are given in the following sections.

A.8.1 Computing Resources

- `Initial-Memory = mem_val`
- `Memory-Increment = mem_val`
- `Max-Memory = mem_val`

There are three parameters which control the amount of memory that *eShell* may use when executing. `Initial-Memory` defines the initial amount of memory that the program will use, and `Memory-Increment` provides an optional memory increment size. If memory is exhausted during execution, and an increment size has been provided, then the memory will be extended by the specified value. This procedure will continue as necessary until `Max-Memory` is reached.

The actual units available to define the memory sizes are described in Chapter 2.

For example:

```
Working-Memory = 100kw
Memory-Increment = 10kw
Max-Memory = 5mw
```

initially allocates 100 thousand words of memory to *eShell*. If additional memory is required, it is added in 10 thousand word increments until the maximum of 5 million words is obtained.

A.8.2 Processing Defaults

- `Tolerance = real_val`

The parameter `Tolerance` is used to define the tolerance used for comparing floating point values while using *eShell*. The value may be overridden using the *eShell* command:

```
SET TOLERANCE TO value [ PERCENT ];
```

as described in Chapter 13 of the *eShell* User's Manual.

A.8.3 I/O System Parameters

- System-Database-Loc = 'db_name'
- Temp-Loc = 'path_loc'

System-Database-Loc specifies the name of the directory where the **eShell** system database resides., and *Temp_loc* defines a location in which the temporary database will reside.

A.8.4 Program Authorization

The following parameters are used for **eShell** program authorization and should only be modified under instruction by UAI.

- ESHPAK01 = 'keydata'
- ESHPAK02 = 'keydata'
- ...

Defines the Program Access Key data that is used to validate your use of the **eShell** program.



Do not change these data unless instructed by UAI. If they are changed, the program will not function.

INDEX

A

- Adobe Acrobat reader 4-5, 5-4, 6-1, 7-2
- ALTER library 4-3
- Analysis output control
 - UAI/NASTRAN** A-12
- applib** configuration parameters
 - Computing resources A-20
- Applications programming
 - Compiler options 6-2
 - Development 6-2
 - Examples 6-4
 - Interface 6-2
 - Using **applib** 6-3
- Assign processing
 - UAI/NASTRAN** A-13
- ASTROS**
 - Applications problem library 5-3
 - astros** script 5-1
 - Default file names 5-2
 - Log file 5-2
 - makelocalastros** script 5-3
 - Modifying the **astros** script 5-3
- ASTROS** configuration parameters
 - Computing resources A-16
 - Data checking A-17
 - Element options A-18
 - I/O system parameters A-18
 - Matrix conditioning A-16
 - Optimization control options A-18
 - Print file controls A-15
 - Program authorization A-20
 - Solution techniques A-17

B

- BLAS library 6-2
- Block size
 - Data component 3-3

- Index component 3-3
- Selecting 3-3

C

- Computer system requirements 1-4
- Computing resources
 - applib** A-20
 - ASTROS** A-16
 - eBase** A-3
 - eShell** A-23
 - UAI/NASTRAN** A-7
- Configuration
 - See also Preference files
- Configuration files
 - Definition 2-1
 - Modifying 2-7
 - Overriding 2-7
 - Overview 2-1
 - Sections 2-6
- Configuration parameters
 - AlterLib-Loc A-13
 - ASTPAK A-20
 - Auto-Assign A-13
 - Auto-Index-Archive A-14
 - AutoSing-MaxCheck A-21
 - AutoSing-MaxRatio A-21
 - AutoSPC-Eps A-8, A-16
 - AutoSPC-Method A-8, A-16
 - AutoSPC-Print A-8, A-16
 - AutoSPC-Select A-8, A-16
 - Beam-Automatic-Warping A-12
 - Beam-Offset-MaxWarnings A-10
 - Beam-Offset-Ratio A-10
 - BulkData-Echo A-7, A-15
 - Constraint-MaxWarnings A-10
 - Constraint-Retention-Factor A-19
 - Constraint-Retention-Lbound A-19
 - Convergence-Limit A-19
 - CPU-Time A-7
 - Data-File-BlockSize A-4
 - Default-Grid-Temperature A-11
 - Diag-Output A-7
 - Dynamic-Initial-Memory A-8, A-20

Dynamic-Max-Memory A-8, A-20
 Dynamic-Memory-Increment A-8, A-20
 eBase-Initial-Memory A-3
 eBase-Max-Memory A-3
 eBase-Memory-Increment A-3
 EBSPAK A-6
 Eigen-Normalization A-11, A-17
 Element-KE-Threshold A-13
 Element-SE-Threshold A-13
 ESHPAK A-24
 External-File-BlockSize A-13
 External-Temp-Loc A-13
 File-Utilization A-6
 FSD-Move-Limit **A-19**
 Hexa-Bubble A-11
 Index-Archive A-14
 Index-File-BlockSize A-4
 Initial-Memory A-23
 K6Rot A-11, A-18
 License A-2
 Linear-Plate-Center-Stress A-12
 Linear-Plate-Corner-Stress A-12
 Lines-per-Page A-6, A-15
 Local-Pref-Default-Name A-3
 Local-Pref-File-Loc A-2
 Logical-Unit-Input A-14
 Logical-Unit-Output A-14
 Manufacturer A-2
 Mass-Orthogonality-Test A-11
 MATPAK A-22
 Max-Index-Building-Memory A-4
 Max-Memory A-23
 Max-Stationary-Obj **A-19**
 Maximum-Filename-Length A-13, A-18
 Maximum-Memory A-7, A-16
 Maximum-Print-Lines A-6, A-15
 Memory-Increment A-23
 Model A-2
 MP-Move-Limit A-19
 NASPAK A-15
 News-File A-7
 NonLinear-Nset-Autospc A-8
 Nonunique-Index-Archive-Attribute A-14
 Optimization-Maxiter A-19
 Optimize-Index-Building A-4
 Optimizer-Select A-19
 Panel-Buckling-Dynamic-Target A-18
 Panel-Buckling-Initial-Terms A-18
 Panel-Buckling-Linearization-Power A-18
 Panel-Buckling-Maximum-Terms A-18
 Panel-Buckling-Term-Select A-18
 Perm-eBase-Data-Loc A-5
 Perm-eBase-Index-Loc A-5
 Perm-eBASE-Integrity A-4
 Physical-Memory A-7, A-16
 Plate-Aspect-Ratio-MaxWarnings A-10
 Plate-Aspect-Ratio-Ratio A-10
 Plate-Geom-Check A-9
 Plate-Geom-Max-Percentage A-10
 Plate-Geom-MaxWarnings A-9
 Plate-Geom-Percentage A-9
 Plate-Offset-MaxWarnings A-10
 Plate-Offset-Ratio A-10
 Plate-Stress-Coordinate A-12
 Quad-Warping-Check A-10
 Quad-Warping-MaxWarnings A-10
 Quad-Warping-Ratio A-10, A-17
 Quad-Warping-Threshold A-12
 SA-Gradient-FDSTEPsize **A-19**

Sequencer-Criteria A-9, A-17
 Sequencer-Method A-8, A-17
 Sequencer-Print A-9, A-17
 Site A-2
 Solid-Geom-Check A-9
 Solid-Geom-Max-Percentage A-9
 Solid-Geom-Percentage A-9
 Solid-Material-Coordinate A-12
 Solid-Stress-Coordinate A-12
 Solver-Order A-22
 Solver-Print A-22
 Solver-Select A-22
 Sparse-Solver-Version A-22
 Stationary-Deltaobj-value **A-19**
 Sys-Pref-Default-Name A-3
 Sys-Pref-File-Loc A-2
 System-Database-Loc A-13, A-18, A-24
 Temp-eBase-Data-Loc A-5
 Temp-eBase-Index-Loc A-5
 Temp-eBASE-Integrity A-4
 Temp-Loc A-24
 Time-I/O A-22
 Time-Loose-Loop A-22
 Time-Matrix-Columns A-22
 Time-Matrix-Strings A-22
 Time-Matrix-Terms A-22
 Time-Tight-Loop A-22
 Tolerance A-23
 Undesigned-Stress-Constraints **A-17**
 Unique-Index-Archive-Attribute A-14
 Upper-Case-Assign A-14
 User-Pref-Default-Name A-3
 User-Pref-File-Loc A-2
 Working-Memory A-7, A-16

D

Data checking
 ASTROS A-17
 UAI/NASTRAN A-9
 Default file names
 UAI/NASTRAN 4-2
 Delivery materials 1-2
 Delivery media 1-5
 Demonstration problem library
 UAI/NASTRAN 4-3
 Dynamic memory management
 ASTROS 5-4
 applib 6-8
 eBase 3-4
 eShell 7-7
 UAI/NASTRAN 4-5

E

eBase
 Dynamic memory 3-4
 File location 3-2
 File naming conventions 3-2
 Logical names 3-2
 Multiple file systems 3-3
 Overcoming file size limits 3-4
 Persistence 3-1

Physical model 3-1
 Physical names 3-2
eBase configuration parameters
 Computing resources A-3
 I/O system parameters A-5
 Program authorization A-6
eBase data component 3-3
eBase index component 3-3
 Element options
 ASTROS A-18
 UAI/NASTRAN A-11
eShell
 Computing resources A-23
 eShell script 7-1
 I/O system parameters A-24
 Online manuals 7-2
 Preference files 7-3
 Processing defaults A-23
 Program authorization A-24
 Tutorial examples 7-2

H

Hardware requirements 1-4
 Host configuration parameters
 Preference override information A-2
 Site description A-2
 HP-GL plotting program 8-3

I

I/O system parameters
 ASTROS A-18
 eBase A-5
 UAI/NASTRAN A-13
 Index archive control
 UAI/NASTRAN A-14
 Installation
 ASTROS 1-2
 eBase 1-3
 UAI/NASTRAN 1-2
 Installation instructions 1-6
 Installation support 1-11
 Interface files
 UAI/NASTRAN 4-13

L

LAPACK library 6-2
 Log file
 ASTROS 5-2
 UAI/NASTRAN 4-2

M

matlib configuration parameters
 Program authorization A-22

Solver options A-21
 Timing constants A-22
 Matrix conditioning
 ASTROS A-16
 UAI/NASTRAN A-8
 Modifying the **nastran** script
 UAI/NASTRAN 4-3
 Motif plotting program 8-4
 MSC/PATRAN interface 4-5

N

NEWS file 4-4

O

Online manuals
 applib 6-1
 ASTROS 5-4
 eShell 7-2
 matlib 6-1
 UAI/NASTRAN 4-5
 Optimization control options
 ASTROS A-18

P

Plot file format 8-11
 PLOT10 program 8-2
 Plotter commands 8-9
 Plotting programs 8-2
 HP-GL 8-3
 Motif interface 8-4
 Postscript 8-2
 Postscript plotting program 8-2
 Preference files
 Components 2-2
 Default 2-1
 Format 2-3
 Overview 2-1
 System 2-1
 User 2-1
 Preference override information
 Host section A-2
 Print file controls
 ASTROS A-15
 UAI/NASTRAN A-6
 Processing defaults
 eShell A-23
 Program authorization
 ASTROS A-20
 eBase A-6
 eShell A-24
 matlib A-22
 UAI/NASTRAN A-15

S

SDRC Dataloader 4-4
 Selecting **eBase** block sizes 3-3
 Site description
 Host section A-2
 Solid-Geom-MaxWarnings A-9
 Solution techniques
 ASTROS A-17
 UAI/NASTRAN A-11
 Solver options
 matlib A-21
 Sparse matrix solvers
 UAI/NASTRAN 4-6

T

Timing constants
 matlib A-22

U

UAI/NASTRAN
 ALTER library 4-3
 Default file names 4-2

Demonstration problem library 4-3
 Dynamic memory management 4-5
 Interface files 4-13
 Log file 4-2
 Modifying the **nastran** script 4-3
 MSC/PATRAN interface 4-5
nastran script 4-1
 NEWS file 4-4
 Online manuals 4-5
 SDRC dataloader 4-4
 Sparse matrix solvers 4-6
 User's guide problem library 4-3
UAI/NASTRAN configuration parameters
 Analysis output control A-12
 Assign processing A-13
 Computing resources A-7
 Data checking A-9
 Element options A-11
 I/O system parameters A-13
 Index archive control A-14
 Matrix conditioning A-8
 Print file controls A-6
 Program authorization A-15
 Solution techniques A-11
 User's guide problem library
 UAI/NASTRAN 4-3