SIEMENS

NX Nastran 10 Installation and Operations Guide

Contents

Proprietary & Restricted Rights Notice

Chapter 1: Introduction

- *Document Scope*
- *Key for Readers*
- *Document Structure*
- *The Directory Structure*

1.1 Document Scope

The NX Nastran Installation and Operations Guide (IOG) provides instructions on how to install, customize, and use NX Nastran on Linux and Windows systems. This document assumes that you have a working knowledge of the applicable operating environments.

1.2 Key for Readers

The IOG uses certain stylistic conventions to denote user action, to emphasize particular aspects of an NX Nastran run, or to signal other differences within the text.

Italics	Represent user-specified variables.	
	Example:	The system RC file is *install_dir*/conf/nastrrc.
Courier font	Indicates system input or output.	
	Example:	`$ install_dir /bin/ugsid`
Quote marks	Distinguish words or phrases such as lowercase keywords, commands, variables, Dbsets or file suffixes from regular text.	
	Example:	If “out” is not specified, NX Nastran saves the output files using the basename of the input data file as a prefix.

1.3 Document Structure

The IOG focuses on three areas of NX Nastran use and also features additional information in the form of appendixes. Chapters 2 and 3, discussing installation and configuration, are the only two chapters intended for system administrators; all other information in this document is intended for NX Nastran users.

Installation and Configuration

Basic and Advanced Use

Chapter 5 presents the basic functions of the nastran command and provides some details on how to use system files and databases. Chapter 6 explains how to use the advanced features of the nastran command and includes information on computer resource management.

Utility and Sample Programs

The final two chapters contain information on utility and sample programs, including NX Nastran Access and the beam server. Chapter 7 focuses on using and customizing utility programs, while Chapter 8 explains how to build and use sample programs.

Supplementary Information

Appendix A contains a glossary of terms. Appendix B reviews keywords and environmental variables. Appendix C details system descriptions (Appendix C is not included with the NX installation).

1.4 The Directory Structure

The installation directory structure provides the following capabilities:

- Multiple versions of NX Nastran products.
- Multiple computer architectures.

  This structure doesn't permit Linux or Windows installations to share the same directory tree, e.g., on an NFS or Samba server. However, installations from a variety of Linux platforms may be shared from the same directory tree on an NFS or Samba server.

Figure 1-1 shows the structure of the *install_dir* directory, which is selected during installation. For the nxn*r* directory, the *r* indicates the version number of NX Nastran.

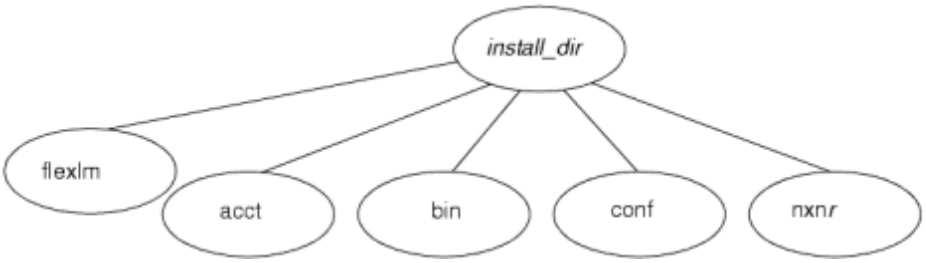


Figure 1-1. Directory for install_dir

Multiple Products Support

The NX Nastran installation directory structure supports multiple products by using product-dependent and architecture-independent directories and files. For example, Figure 3-1 shows that the *install_dir*/nxn*r*/nast directory on Linux and *install_dir*\nxn*r*\nast on Windows contains the product-dependent files for NX Nastran while the util and access directories contain the product-independent files for the various utilities and NX Nastran ACCESS.

Multiple Computer Architecture Support

The NX Nastran installation directory structure also supports multiple computer architectures by using architecture-dependent directories and files. All files that are dependent upon a computer architecture are isolated in a single architecture directory *install_dir*/nxn*r*/*arch* on Linux and *install_dir*\nxn*r**arch* on Windows, where *arch* is the name of the architecture. See Table 3-1).

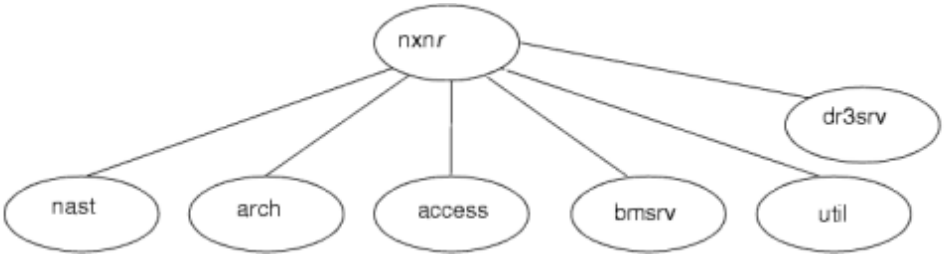


Figure 1-2. Directory for nxnr

The *install_dir*/nxn*r*/nast directory on Linux and *install_dir*\nxn*r*\nast directory on Windows contains news, documentation, and sample problems for NX Nastran. None of these files is architecture dependent.

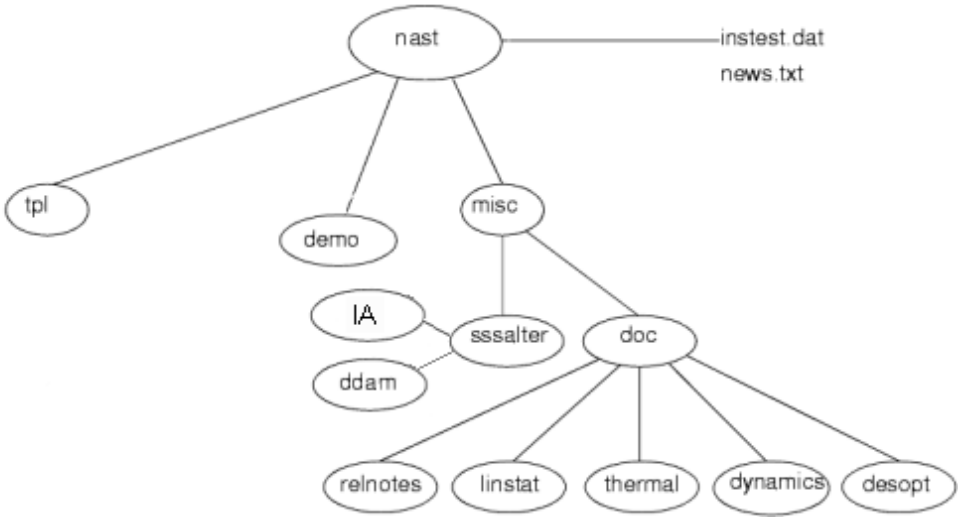


Figure 1-3. Directory for nast

The NX Nastran ACCESS directory (*install_dir*/nxn*r*/access on Linux and *install_dir*\nxn*r*\access on Windows) contains source and make files for the NX Nastran ACCESS sample programs (see Figure 1-4). None of these files is architecture dependent. The DBIO library, which is architecture dependent, is located in the architecture directory, i.e., *install_dir*/nxn*r*/*arch* on Linux and *install_dir*\nxn*r**arch* on Windows.

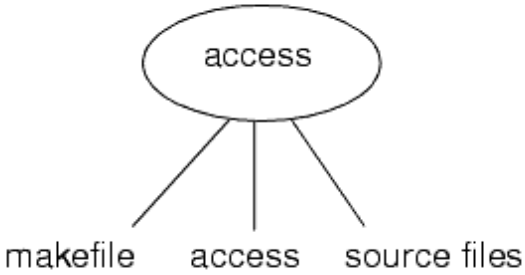


Figure 1-4. Directory for access

The beam server directory (*install_dir*/nxn*r*/bmsrv on Linux and *install_dir*\nxn*r*\bmsrv on Windows) contains source and make files (see Figure 1-5) for the beam server sample programs. None of these files is architecture dependent. The beam server library, which is architecture dependent, is located in the architecture directory, i.e., *install_dir*/nxn*r*/*arch* on Linux and *install_dir*\nxn*r**arch* on Windows.

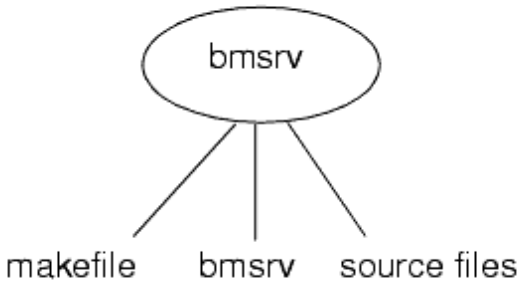


Figure 1-5. Directory for bmsrv

The dynamic response server directory (*install_dir*/nxn*r*/dr3srv on Linux and *install_dir*\nxn*r*\dr3srv on Windows) contains source and make files (see Figure 1-6) for the dynamic response server sample programs. None of these files is architecture dependent.

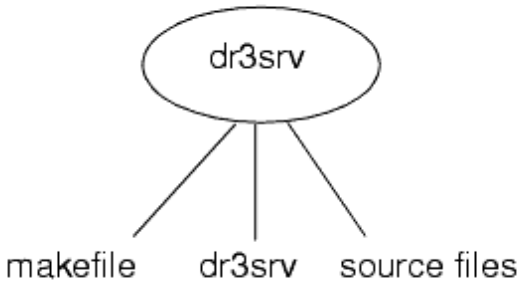


Figure 1-6. Directory for dr3srv

The utility programs directory (*install_dir*/nxn*r*/util on Linux and *install_dir*\nxn*r*\util on Windows) contains source and make files (see Figure 1-7) for the utilities that are also delivered in source form. None of these files is architecture dependent.

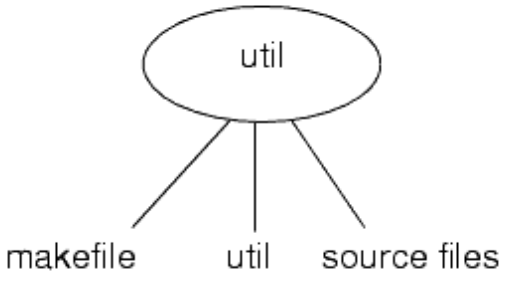


Figure 1-7. Directory for util

Chapter 2: Installing NX Nastran

- *Installation Overview*
- *Installing NX Nastran on Windows Systems*
- *Installing NX Nastran on Linux Systems*
- *Configuring the NX FLEXlm License Manager*

2.1 Installation Overview

Before installing NX Nastran, you must ensure that your computer and operating system meet the requirements of the product. The product system requirements can be found in the README.txt file in the top level directory of the NX Nastran DVD distribution media.

- Windows Installation

  When you load the DVD on a Windows machine, the autoplay capability should automatically open the NX Nastran Installation page. If it doesn't, use Windows Explorer to open the drive containing the installation media. Then, double-click AutoPlay.exe. The installation page includes options to install the license server, NX Nastran, the documentation server, and the documentation.

  Alternately, the following separate programs are included on the NX Nastran DVD distribution media.

  - NX Nastran setup programs are contained in the *I386\NXNastran* and *I386\NXNastran64* directories.
  - The NX FLEXlm License Manager setup program is contained in the *I386\FLEXlm* directory. See Configuring the NX FLEXlm License Manager.
  - The documentation installation instructions are in documentation_install.pdf.

- Linux Installation
  - The interactive NX Nastran installation script, *nxnsetup*, uses a series of menus to guide you through the installation procedures. Alternately, the /nxn*r* directory contains subdirectories for Linux. The subdirectories contain platform-specific base.tar and util.tar files.
  - To install the license manager on Linux, you use the setup program *SPLMLicenseServer_v5.3.1_linux_setup.bin*, which is contained at the top level of the installation media.

    See Configuring the NX FLEXlm License Manager.
  - The documentation installation instructions are in documentation_install.pdf.
  - To run NX Nastran on any Linux machine, you must have the system library libnuma.so.1 installed on your operating system. See Linux library requirement.

Additional Considerations

- Use the Proper Privileges

  To install any program contained on the distribution media, the account with which you install the programs must have the necessary administrative privileges to allow you to install software. If you are unsure as to whether your account has the necessary privileges, contact your system administrator.

- Verifying the Required Disk Space

  Depending on the product features you install, NX Nastran can use varying amounts of hard disk space. This amount will vary depending upon the platform on which you're installing NX Nastran

and the features you choose to install. The installation program gives detailed information about the required disk space.

In addition to the amount of disk space required for product installation, you should also ensure that you have adequate space in which to store the files generated by NX Nastran.

- Temporarily Disable Anti-virus Products

  Some anti-virus products will generate “false positives” when you install software packages. This may interfere with the correct installation of the product. All anti-virus packages, at a minimum, slow down the transfer of files from the installation disk to the target disk. For these reasons, you should temporarily disable any anti-virus products on the computer on which you’re installing NX Nastran.

2.2 Installing NX Nastran on Windows Systems

To install and run NX Nastran on a Windows system, you must verify your system meets the requirements of the product. The product system requirements can be found in the README.txt file in the top level directory of the NX Nastran DVD distribution media.

Since NX Nastran executables are now all 64-bit, you must run NX Nastran on a 64-bit machine. See LP-64 and ILP-64 Executable Information.

To build the NX Nastran Utility programs, your Windows system must also have a suitable set of compilers. Refer to “Using the Utility Programs” for details.

Installation Procedure for Windows Systems

1. Load the Installation DVD

2. Use Autoplay to Begin the Installation

   When you load the DVD on a Windows machine, the autoplay capability should automatically open the NX Nastran Installation page. If it doesn’t, use Windows Explorer to open the drive containing the installation media. Then, double-click AutoPlay.exe.

3. On the NX Nastran Installation screen, select the product to install.

   - The software installation option.
   - The FLEXlm license server installation option
   - The Siemens PLM documentation server installation option.
   - The documentation installation option.

   Once you’ve selected the product to install, select Next on the Welcome screen of the installation program.

4. Read and Approve the License Agreement

   Read the displayed license agreement on the *License Agreement* screen. If you do not accept the agreement, the NX Nastran installation program will not allow you to continue the installation.

5. Enter the Appropriate Customer Information

   On the *Customer Information* screen:

   - Enter your name in the *User Name* field as the registered user of the product.
   - Enter the name of your company in the *Organization* field.
   - Use the *Install this application for:* option to specify how you want to install the product.
     - Select *Anyone who uses this computer (all users)* to allow all users to see the NX Nastran icon in your computer's start menu.
     - Select *Only for me* to prevent other users of your machine from seeing the NX Nastran icon.

6. Select the Installation Setup Type

   The *Setup Type* screen allows you to specify the type of NX Nastran installation to perform.

Complete	Installs all directories, including the input file examples in the Test Problem Library.
Custom	Lets you select whether you want to install the input file examples in the Test Problem Library. In some cases, where disk space is at a premium, you may choose not to install these examples.

   If you select a *Custom* installation, you can:

   - update your installation at a later date to add any of the features that you didn't select during the initial installation
   - pick *Change* from the *Custom Setup* screen to change the installation path for the software.

7. Select the Destination Folder

   Use the *Destination Folder* screen to specify the location of the installation folder (*install_dir*) for the selected product. Pick *Change* to change the location of the destination folder.

8. Select the License Server

   Specify the name of the computer that you will use to obtain a license for NX Nastran. This entry will define the *authorize* keyword in the runtime configuration file. You can optionally redefine this keyword after the installation is complete. See "Customizing the Runtime Configuration Files".

   The format of the *authorize* keyword is *port@hostname* where *port* is the port in use for the license server and *hostname* is the name of the license server machine. You only need to enter the *hostname* in this field, and the installation program will include the default port, 28000.

   See Selecting the License Server.

9. Begin the Installation

   Pick *Install* on the *Ready to Install the Program* screen to begin the installation. You can cancel the installation of the product at anytime without adverse effects to your system. If you cancel the installation, the Windows Installer service ensures that any changes made to your computer are rolled back.

10. Complete the Installation

    When the installation is complete, pick *Finish* to exit the installation program. If both NX Nastran and the license management software installed correctly, NX Nastran is ready to run.

11. Additional Considerations

    The default for the NX Nastran scratch file directory is “%TEMP%”. Having this directory on a separate drive from the system swap file can help performance. You can use the nastr runtime configuration file (nastr.rcf) to change the location of the scratch file directory. See “Customizing the Runtime Configuration Files”.

    To run NX Nastran from any directory, you can add *install_dir*\bin to your system PATH.

2.3 Installing NX Nastran on Linux Systems

The interactive installation script, nxnsetup, uses a series of menus to guide you through the installation procedures. It also uses the standard gunzip and wget utilities from the Free Software Foundation.

The nxnsetup script is a fairly complex Korn shell script. If too many processes are running when nxnsetup runs, the script may hang or generate utility errors. For best results, close or exit other applications before running nxnsetup.

Note

To run nxnsetup, you must have the Korn shell available as /bin/ksh.

The following environment variables affect nxnsetup:

- NXN_*ARCH*
- NXN_*BASE*
- NXN_*CMDSUB*
- NXN_*SETUP*
- *TMPDIR*

See “Environment Variables” . More information on the effect of these environment variables on the installation script can be found in the nxnsetup shell script.

Note

Any run time libraries needed by NX Nastran are included in this distribution.

1. Log on to the system with proper privileges and mount the drive which will contain the installation DVD media.

2. Optionally run nxnsetup info to check system information

   Before you use nxnsetup to initiate the installation, you may want to use the nxnsetup info tool to check the configuration of your system. For example,

```
/mnt/cdrom/nxnsetup info
```

3. Run nxnsetup to begin the installation

   How you initiate the nxnsetup installation script depends on whether you're installing the product from a local or remote CD-ROM. For example,

   ```
   /mnt/cdrom/nxnsetup
   ```

4. Confirm basic system information and begin the installation

   On the *Introduction* screen, the software displays basic information about your system and the installation. If the software incorrectly identifies your system, exit the script. Before restarting the script, set the environment variable NXN_ARCH to the correct architecture name as shown in Table 3-1.

5. Select the base installation directory

   From the *Installation Base Directory* screen, enter the path for the directory in which you want to install NX Nastran, such as:

   ```
   /siemens/nxnastran
   ```

   If that directory doesn't already exist, the software asks whether you want it to automatically create that directory.

6. Select the type of installation to perform

   From the *Installation Type* screen, select the type of installation that you want to perform. This screen also lists the amount of disk space required by each installation type.

   NX Nastran Installation Types:

   Full (default) Installs all directories, including the advanced examples in the Test Problem Library, and utilities.

   Minimum Installs only the minimum number of directories needed to run the product. Does not install the beam server source files, the dr3 server source files, the utilities in the *install_dir*\nxn*r*\util directory, the advanced examples in the Test Problem Library. Does not install common utilities, such as receiver, smplr, tabtst, trans, neutrl, plotps, or qaprt. Also does not install any of the files in the *install_dir*\nxn*r*\nast\demo directory (demonstration problems) or any files in the *install_dir*\nxn*r*\nast\misc directory (includes sssalters and documentation files).

   Custom Installs NX Nastran directories for multiple architectures on one server. Designed for environments in which one single server is designated to serve the NX Nastran software to different platforms.

Note

The disk space requirements displayed on the *Installation Setup* screen don't include the scratch space needed to decompress the installation files. Depending on the type of installation, up to 45 MB of additional space may be needed in the installation file system or the temporary file system. The temporary file system is defined by the:

- -t option on the nxnsetup command line, for example:

  `mnt/cdrom/nxnsetup -t alternate_temporary_directory`

- TMPDIR environment variable

- default temporary directory, e.g., /tmp.

7. Select the architectural components to install (custom install type only)

   If you selected *Custom* on the *Installation Type* screen, use the options on the *Architectural Components* screen to select the operating system-specific NX Nastran program and utility executable to install.

8. Select the optional components to install (custom install type only)

   If you selected *Custom* on the *Installation Type* screen, use the options on the *Optional Components* screen to select which NX Nastran components to install.

9. Verify the selected installation information

   Verify your selections and make any necessary changes before proceeding.

10. Examine the system configuration report

    The software automatically generates a report that lists detailed information about the system on which you're installing NX Nastran. The software displays a warning if any of the configuration items, such as the amount of temporary disk space, don't meet the installation requirements. You can choose to either abort or continue the installation (although the installed products may not function correctly until the failure is corrected).

11. Select the products to configure

    Use the options on the *Configure Products* screen to select which of the products you've selected for installation to configure.

12. Specify whether to use configuration files from a previous installation

    If you have a previous version of NX Nastran installed either on your system or in an NFS-accessible path, the nxnsetup tool can use information in the system-wide configuration files for that installation to establish default values for this installation.

13. Select product configuration parameters

    Next, select the configuration parameters for each of the products which you've chosen to configure.

Table 2-1. NX Nastran Configuration Options

Option	Configuration
License server	Enter the number of the port (28000) and name of the license server. See Selecting the License Server.
Run installation test	Specify whether to have the software test the installation. The software only performs the installation test on the current architecture.
Create version links	If you have multiple versions of NX Nastran installed, creates symbolic links between those files.
Create PATH links	Changes the system path directory to point to the *install_dir*/bin directory.
sdirectory	Enter the path for the NX Nastran scratch files.
buffsize	Specify the buffer size (words) used for I/O transfer for each DBset. See “Suggested BUFFSIZE Values”.
memory	Either specify the amount of memory to use or enter “estimate” to have the software use the NX Nastran estimate utility to estimate memory requirements. See “Determining Resource Requirements”.

14. Review the product licensing requirements

    Use the *Product Licensing Requirements* screen to review the selected licensing methods and licensing installation status for each product you’re installing.

15. Review the installation options and initiate the installation

    Use the *Installation Review* screen to briefly review the options you’ve selected during the installation program, such as the path for the installation directory and the list of the products you’re installing.

    If all the displayed information is correct, enter Y to begin installing the selected products. During the installation, the software creates two separate files: nxnsetup.pbk and nxnsetup.log.

    - The nxnsetup.pbk or “playback” file” records all of the options you selected on the various nxnsetup screens. If you need to install NX Nastran on multiple, identical machines, you can use the playback file to streamline the installation process.

    - The nxnsetup.log or “log” file records all the activity of the nxnsetup installation program, including any errors or problems encountered during the actual installation. You can use the log file to verify the installation.

Linux library requirement

Beginning with NX Nastran 9.1, to run NX Nastran on any Linux machine, you must have the system library libnuma.so.1 installed on your operating system. Libnuma.so.1 is distributed in the following packages.

- For Redhat EL, the package is numactl.

- For SuSE Enterprise, the package is libnuma1.

Contact your operating system vendor for information. If the libnuma.so.1 library is not installed on your Linux system, NX Nastran will exit immediately with the following message in the log file.

```
analysis: error while loading shared libraries:libnuma.so.1
cannot open shared object file: No such file or directory
Command exited with non-zero status 127
```

Repeating a Linux Installation

You can repeat any installation using the playback file (nxnsetup.pbk) that's automatically generated during every installation. You can use a playback file to reinstall NX Nastran on the same computer or to make an identical installation on another computer. The following command is used:

```
nxnsetup playback-file
```

where *playback-file* is the playback file generated during a previous installation (the default playback file is *install_dir*/nxnsetup.pbk).

When you use a playback file during an installation:

- the architecture of every computer using the playback file must be the same as the architecture of the computer that generated the playback file
- you cannot change any installation options, such as the selected products or installation types
- if the playback file was generated during an installation from a remote CD-ROM, and you want to mount the CD-ROM in a different system when using the playback file, you can change the node and user with the "-r" option. For example:

  ```
  nxnsetup -r node [-m /CDROM] playback-file
  ```

  or

  ```
  nxnsetup -r user@node [-m /CDROM] playback-file
  ```

  You only need to specify the -m option if the CD-ROM mount point changed.
- you can change the installation base directory by specifying the -b option, for example:

  ```
  nxnsetup -b new-install-base playback-file
  ```

2.4 Configuring the NX FLEXlm License Manager

To use NX Nastran, you must install the Siemens PLM License Server. It must be either installed on the local computer in which you will run NX Nastran, or onto a computer that is accessible via TCP/IP to your network.

All of the installation instructions for the license server are included in *SPLM_Licensing.pdf*, which is contained at the top level of the installation media. The Licensing User Guide *SPLMLicensing_user_guide.pdf* is also included in the same location on the installation media.

- To install on Windows, you can either run *AutoPlay.exe*, or run the setup programs contained in the *I386\FLEXlm* directory.
- To install on Linux, you run the setup program *SPLMLicenseServer_v5.3.1_linux_setup.bin*, which is contained at the top level of the installation media.

Selecting the License Server

Once you have successfully installed the license manager using the instructions in *SPLM_Licensing.pdf*, you must point NX Nastran to the license server.

When you choose a computer to use as the license server for your NX Nastran licenses, you should ensure that the operating system's product license allows you at least as many specific, simultaneous connections as the number of product licenses you have. Some operating systems, such as Microsoft Windows, restrict the number of allowable simultaneous connections, through software management or license agreements.

Methods of Selecting the License Server

To determine your license server, NX Nastran uses the first non-null value that it finds in the following hierarchy:

1. The NXN_LICENSE_FILE environment variable.
2. The value of the *authorize* (or "auth") keyword on the command line when you run NX Nastran.
3. The value of the *authorize* keyword in an RC file (nastr.rcf on windows platforms, nastrrc on Linux platforms).
4. The value of the SPLM_LICENSE_SERVER environment variable.
5. The value of the UGS_LICENSE_SERVER environment variable.

For more information on configuring the *authorize* keyword in the runtime configuration file, see "Customizing the Runtime Configuration Files".

Identifying the Licensing Source with the Authorize Keyword

With NX Nastran, the authorize keyword indicates the licensing source. The value for authorize can be any of the following:

Value	Comments	Example
@node	The specified node is the license server using the default port number 28000. This form cannot be used in a list or quorum specification.	`auth=@troll` where node “troll” is a FLEXlm license server using the default port number
port@node	The specified node is running a license server listening on the specified port.	auth=28000@troll where node “troll” is a FLEXlm license server using the port number 28000
value,value,value	A quorum of three FLEXlm license server nodes.	
value:value:...	Linux: A list of FLEXlm license server nodes, or quorums. Note: In this case, you must explicitly specify the port number.	`auth=28000@banana1:28000@banana2` where two alternate network license servers, “banana1” and “banana2,” will be used to provide network licensing services
value;value;...	Windows: A list of FLEXlm license server nodes, or quorums.	

Understanding Authorization Keyword Processing

- If the software finds non-null value for the authorize keyword, it starts your NX Nastran job.
- If the software can’t find a non-null value cannot be found for authorize, the software issues the following User Fatal Message (UFM) when you use the nastran command:

```
*** USER FATAL MESSAGE (nastran.validate_authorize)
    authorize=""     (program default)
 The keyword shall not be blank or null.
```

- If the software later determines that the information you specified with authorize is either invalid or insufficient for the analysis you’re performing, it prints a UFM 3060 error message in the associated .f06 file:

```
*** USER FATAL MESSAGE 3060, SUBROUTINE MODEL - OPTION opt NOT IN APPROVED LIST.
     SYSTEM DATE (MM/DD/YY): mm/dd/yy
     SYSTEM UGSID: d (DECIMAL) h (HEXADECIMAL) SYSTEM MODEL NUMBER: m, SYSTEM OS CODE: c
```

  where *opt* is a keyword indicating the specific capability requested. The initial authorization check is for option “NAST”, subsequent checks request specific features as required by your job.

- When NX Nastran attempts to run, but cannot because a license is unavailable, there is an option to have the job retried automatically. NX Nastran will retry the job every minute up to the value of the AUTHQUEUE keyword. You can define the AUTHQUEUE keyword in the runtime configuration file. See “Customizing the Runtime Configuration Files”.

Chapter 3: Configuring NX Nastran

3.1 Overview

This chapter is intended for system administrators or anyone who needs to manage an NX Nastran installation. It starts with information on tuning your system for better performance. Other items that may require configuration include system resource limits, the command initialization file, runtime configuration files, timing blocks, and queue commands.

Documentation Conventions

Two documentation conventions are used throughout the remainder of this document (typically in directory specifications):

- The string "*install_dir*" indicates the directory where NX Nastran was installed. This might be "/siemens", and on Windows "c:/siemens".
- Throughout this document, while file pathnames and sample commands for Windows systems will use the standard backslash "\" directory separator character, NX Nastran also accepts pathnames using the slash "/" character as a replacement
- The string "*arch*" indicates the architecture name for your computer.

NX Nastran Architecture Names

The architecture names are as follows:

Note

Note: Each of the ILP-64 architectures ends with a lower case L.

Table 3-1. NX Nastran Architecture Names

Computer	Executable Type	*arch*
Intel Windows	LP-64	emt64nt
	ILP-64	emt64ntl
X86_64 Linux	LP-64	x86_64linux
	ILP-64	x86_64linuxl

3.2 System-Specific Tuning

This section presents some information on system-specific tuning that can help NX Nastran performance. Additional tuning information may be available in the "Read Me" file

`install-dir\nxnr\readme.txt`.

All Systems

All systems benefit from ensuring the I/O system is configured for the highest possible bandwidth. Setting up disk striping, or RAID-0, for use with NX Nastran databases is one of the most effective I/O performance improvements that can be made for NX Nastran.

Linux

Linux Performance Recommendations

Configuring a high-performance /scratch partition on Linux can be very beneficial. Two or more large SCSI disk drives (>16Gb) can be configured as a RAID0 array, in which each drive is connected to a different SCSI controller (ideally U320). Equally important is the filesystem created on the /scratch partition. Extensive testing has shown that best performance is obtainable with the XFS filesystem, which has been mounted to the /scratch partition with the following settings:

biosize=16,logbufs=8,logbsize=32768,noatime,nodiratime

Linux Power Savings Software

Some Linux machines use a power savings software which will clock down the cpu when it determines that the cpu is under utilized. During this “clock down”, NX Nastran becomes inefficient and solution times increase.

Run the following command to quickly determine if power savings software is installed:

rpm -qa | grep pow

This command will return a non-empty string if a power savings package is installed. If so, ask the system administrator to remove it, or login as system administrator and type the command:

rpm -e `rpm -qa | grep pow`

I/O Performance Libraries for X86_64 Linux

MIO Library

IBM provides an asynchronous library called “MIO (modular I/O)” for x86_64 Linux platform. The MIO library addresses the tuning needs for I/O by using a caching mechanism. This caching mechanism bypasses the operating system I/O caching or buffering. This is especially useful for I/O intensive applications such as NX Nastran.

The keyword “mio” can be used to activate MIO:

- Specify “mio=yes”, and
- Set either (a) “MIO_TOOLS_DIR” environment variable to point to the location of the mio tools directory, or (b) set the environment variable MIO_LIBRARY_PATH to point to the appropriate MIO library directory.

By default, NX Nastran sets the environment variable, MIO_FILES to the following value

```
"*SCR*:*scr* [trace/stats/mbytes | pf/cache=2g/page=2m/pref=2 | trace/stats/mbytes | async/nthread=2/
```

Here “2g” specifies that “2GB” cache size needs to be provided for I/O caching. The files that are cached are “SCR*” and “scr*” files.

You can override the cache size and the page size by explicitly setting the environment variable “MIO_FILES” where only these parameters are varied. Varying other parameters would require knowledge about the I/O system and typically should not be varied without consulting IBM.

Alternatively, you can launch NX Nastran through a wrapper script. For example, to run NX Nastran using a korn-shell, the following script may be used.

```
#!/bin/ksh -x
```

```
#
export MIOSTATS=$MIO_TOOLS_DIR/bin/miostats
#
export MIO_LIBRARY_PATH=$MIO"/BLD/xLinux.64/lib"
export LD_PRELOAD=$MIO_LIBRARY_PATH/libtkio.so"
export TKIO_ALTLIBX="analysis[$MIO/BLD/xLinux.64/lib/get_mio_ptrs_64.so/abort]"

 export MIO_STATS="./%{PROGRAM}.%{PID}.stats"
 export MIO_FILES="*.SCR*:*scr* [\
     trace/stats/mbytes |\
     pf/cache=2g/page=2m/pref=2 | \
     trace/stats/mbytes|\
     async/nthread=2/naio=100/nchild=1 |\
     trace/stats/mbytes]"

nastran foo.dat mem=1gb
```

FFIO Library

SGI provides an asynchronous library called “FFIO” for the x86_64 chip set. The FFIO library addresses the tuning needs for I/O by using a caching mechanism. This caching mechanism bypasses the operating system I/O caching or buffering. This is especially useful for I/O intensive applications such as NX Nastran. To run NX Nastran in the context of FFIO, the following needs to be done.

- Create a script. For example, let the script name be “nxffio”. The contents of which should be as follows (this needs to be modified if wanting to run under csh or other shells):

  ```
  #!/bin/ksh -x
  export LD_PRELOAD=/usr/lib/libFFIO.so
  export FF_IO_OPTS=’*SCR*(eie.direct.diag.mbytes:1024:1640:2:1:1:0)’
  $NXN_BASE/$NXN_VERSD/$NXN_ARCH/analysis
  ```

- Change the permission on this file to add “execute” privilege.

- Run NX Nastran with the command line option “exe=./nxffio”. That is, “nastran foo.dat exe=./nxffio”. Any other FFIO environment can be set either inside the script or at the time of login or on command prompt.

Intel

NX Nastran makes very high memory bandwidth demands, and particular attention should be paid to the memory subsystem. A faster memory bus is more important to NX Nastran performance than a faster processor with a slower memory bus.

Hyper-Threading on Intel Processors

Some Intel processors provide a Hyper-Threading feature in which a single processor can support multiple instructions, thus emulating additional processor cores. When Hyper-Threading is enabled, it can lead to a conflict with threading within NX Nastran for both SMP and DMP executables. Hyper-Threading should be disabled, which can be done permanently through BIOS operations.

Windows Server

By default, Windows Server is configured to cache files as much as possible. This can cause an NX Nastran job to appear to “hang” a system running Windows Server.

To correct this problem, open the “Network” Control Panel applet and select the “Services” tab. Highlight “Server” and push the “Properties” button. Make sure the “Maximize Throughput for File Sharing” radio button is not selected (this is the default). Instead select either “Balance” or “Maximize Throughput for Network Applications”. Changing this option will require you to restart Windows.

Windows I/O Option

The I/O option on Windows, File Mapping or native I/O, is determined automatically by the software. File Mapping is used for Windows XP yet all other Windows systems use a native I/O. In a few cases, I/O on Windows XP can be improved by turning File Mapping off. To turn File Mapping off, include the keyword entry “sysfield=mapio=no” either on the command line, or in the RC file.

3.3 Using the “nxnr” Command

The “nxn*r*” command (where *r* is the version of NX Nastran that you’re using) is shown as a prefix for most of the programs and commands described in this document, for example:

```
nxnr nastran ...
```

By ensuring the nxn*r* command is in each user’s PATH, all the commands and utilities in this release are uniformly available. The nxn*r* command also permits version-dependent utilities, such as TRANS, to be easily accessed.

The nxn*r* command is located in

install-dir\bin\nxn*r*.

3.4 Using the “ugsinfo” Command

The “ugsinfo” command is available to display various hardware and software configuration info. You run this utility with the command

```
nxnr ugsinfo
```

ugsinfo displays a hardware and software configuration report that includes:

- Hostname.
- UGSID.
- Computer Manufacturer.
- OS Name, version, and patches.
- Computer Model.
- Processor type, number, and speed.

- Window manager, Motif version, and graphics board.
- Physical and virtual memory sizes.
- Temporary directory sizes.
- Local disk sizes.

Due to the machine-dependent nature of the information, the report varies between computer architectures.

Note

Root access is required to generate the complete report on some systems. If you are not root when ugsinfo is run, those items requiring root access will be noted in the report.

3.5 Activating NX Nastran Accounting

NX Nastran provides a simple accounting package that collects usage information from each job and saves a summary of the job in the accounting directory, i.e., *install_dir*\acct.

Note

Users must be able to read, write, and create files in the accounting directory.

To activate NX Nastran accounting, set the keyword “acct=yes” in any RC file or on the command line. Placing the keyword in the system wide RC file, *install_dir*/conf/nastr, will enable accounting for all jobs.

Instructions for generating usage summaries from the accounting data are provided in the section titled “Using the Basic Keywords”.

Enabling Account ID and Accounting Data

The “acid” and “acdata” keywords are supported by the nastran command to provide hooks for a site to track additional accounting data. The “acid” keyword may be used to specify an account ID. The “acdata” keyword may be used to specify any additional accounting data needed by a site.

These keywords are activated as follows:

1. Activate accounting by putting the line “acct=yes” in the command initialization file or a system RC file.
2. The account validation keyword, “acvalid”, can be used to validate the “acid” keyword. If “acvalid” is not defined in the command initialization file, NX Nastran will not require the “acid” keyword; if the “acvalid” keyword is defined, NX Nastran will require a valid “acid”. See “Enabling Account ID Validation” for a complete description of this capability.

Enabling Account ID Validation

Account ID validation is enabled by defining a non-null value for the “acvalid” keyword in the command initialization file. “Customizing the Command Initialization File” contains additional information. There

are two types of account ID validation available. The nastran command's built-in regular expression facility can be used if the account ID can be described by a regular expression (see "Using Regular Expressions"). Otherwise an external program can be used.

Validating an Account ID with a Regular Expression

To use a regular expression, the first character of the "acvalid" value must be "f" or "w" and the remainder of the value is the regular expression. The "f" indicates that an "acid" value that is not matched by the regular expression is a fatal error, while "w" indicates that an unmatched value is only a warning. Note, the regular expression is always constrained to match the entire account ID string.

For the following examples, assume "acvalid=f" was set in the initialization file and an account ID is not defined in any RC file.

```
nxnr nastran example
```

This job will fail with a message indicating an account ID is required.

```
nxnr nastran example acid=123
```

This job will be permitted to start. Since a regular expression was not defined, any non-null account ID is valid.

For the following examples, assume "acvalid=w" is set in the initialization file and an account ID is not defined in any RC file.

```
nxnr nastran example
```

A warning message will be issued indicating an account ID is required, but the job will be permitted to start.

```
nxnr nastran example acid=123
```

This job will be permitted to start. Since a regular expression was not defined, any non-null account ID is valid.

For the following examples, assume the following line is set in the command initialization file and an account ID is not defined in any RC file:

```
acvalid=f[A-Za-z][0-9]\{6\}
```

This regular expression requires the account ID to be composed of a single upper- or lower-case letter followed by six digits.

```
nxnr nastran example
```

This job will fail with a message indicating an account ID is required.

```
nxnr nastran example acid=123
```

This job will fail with a message indicating the account ID is not valid.

```
nxnr nastran example acid=Z123456
```

This job will be permitted to start.

Validating an Account ID with an External Program

To use an external program, the first character of the “acvalid” value must be a grave, “‘” and the remainder of the value is a simple command to execute the external program. The command may include keyword references but must not include pipes or conditional execution tokens.

The program must examine the account ID and write zero or more lines to its standard output indicating the result of the examination. A null output indicates a valid account ID. The non-null output is composed of two optional parts. The first part is indicated by an equal sign “=” as the first non-blank character. If this is found, the next blank delimited token is taken as a replacement account ID. With this, the external program can replace the user’s account ID with any other account ID. The second part is indicated by an “f” or “w” character. If either of these two characters are present, the remainder of the line and all remaining lines of output are taken as the body of an error message to be issued to the user. If no message text is provided, but the “f” or “w” are present, a generic message is written.

Before we discuss the external program, let’s first consider some examples of the external program’s output.

```
 =Z123456
```

This job will be permitted to start after the account ID is silently replaced with “Z123456”.

```
 f
The account ID is not valid.
See your Program Manager for a valid account ID.
```

This job will fail with the above message.

```
= Z123456
w
The account ID is not valid, it has been replaced by the standard overhead charge.
See your Program Manager for a valid account ID.
```

This job will be permitted to start after the account ID is replaced with “Z123456” and the above warning message is issued.

Sample Account Validation Programs

The account validation program can be written in any language that can process the command line. Two samples have been provided below.

Note

You must have Perl installed on your system to use the Perl sample account validation program.

The Korn shell version is:

```
#!/bin/ksh
#
# Sample site-defined account validation program.
#
# usage: ksh checkac.ksh _account_file_ _account_id_
#
# If the file containing the list of valid account ID's is not specified
# or cannot be opened, report a fatal error.
#
if [[ $#argv -lt 1 || $#argv > 2 ]] ; then
 print "f"
 print "Illegal usage.  See System Administrator."
elif [[ ! -r $1 || ! -s $1 ]] ; then
 print "f"
 print "Account data file \"$1\" cannot be opened."
 print "See System Administrator."
#
# If no argument is specified, issue a warning and use the default
# account ID of Z123456
#
elif [[ -z $2 ]] ; then
 print "= Z123456"
 print "w"
 print "An account ID has not been specified."
 print "The standard overhead charge has been assumed."
 print "See your Program Manager for a valid account ID."
else
#
#   The file is organized with one account ID per line.
#   Make sure the account ID is in the file.
#
 acid=$(fgrep -ix $2 $1 2>/dev/null)
 [[ -n $acid ]] && {
  print "$acid"
  exit
 }
#
# If we get here, the account is invalid.
#
 print "f"
 print "The account ID is not valid."
 print "See your Program Manager for a valid account ID."
fi
```

The Perl version is:

```
#!/usr/local/bin/perl
#
#
# Sample site-defined account validation program.
#
# usage: perl checkac.pl _account_file_ _account_id_
#
# If the file containing the list of valid account ID's is not specified
# or cannot be opened, report a fatal error.
#
if( $#ARGV < 0 or $#ARGV > 1 ) {
 print "f\n";
 print "Illegal usage.  See System Administrator.\n";
} elsif( ! open AC, $ARGV[0] ) {
 print "f\n";
 print "Account data file \"$ARGV[0]\" cannot be opened.\n";
 print "See System Administrator.\n";
#
# If no argument is specified, issue a warning and use the default
```

```
# account ID of Z123456
#
} elsif( $#ARGV < 1 ) {
 print "= Z123456\n";
 print "w\n";
 print "An account ID has not been specified.\n";
 print "The standard overhead charge has been assumed.\n";
 print "See your Program Manager for a valid account ID.\n";
} else {
#
#   The file is organized with one account ID per line.
#   Make sure the account ID is in the file.
#
 $acid = lc "$ARGV[1]";
 while( $line =  ) {
  chomp $line;
  if( $acid eq lc "$line" ) {
   print "= $line\n";
   exit
  }
 }
#
#   If we get here, the account is invalid.
#
 print "f\n";
 print "The account ID is not valid.\n";
 print "See your Program Manager for a valid account ID.\n";
}
```

On Windows, this program is activated with the following

```
acvalid=`perl install-dir\bin\checkac.pl install-dir\acct\account.dat %acid%`
```

Securing the Accounting ID Settings and Files

To secure the account ID settings, you must set the account ID keywords in a write-protected file and lock the values to prevent changes. For example, the following keywords can be set in the command initialization or system RC file

```
acct=yes
 lock=acct
 lock=accmd
 acvalid=some-value-appropriate-to-your-site
 lock=acvalid
```

You can also use UNIX commands if they are supported on your system to secure the accounting files to prevent unauthorized modification or inspection of the accounting data. This can be done by making the accounting logging program, *install_dir*/nxn*r*/*arch*/acct, a “set uid” program.

Note

Before making *install_dir*/nxn*r*/*arch*/acct a set-uid program, you should carefully review the *install_dir*/nxn*r*/util/edsact.c source code, ensure that you have built *install_dir*/nxn*r*/*arch*/acct in a controlled and repeatable manner, and have performed adequate testing to ensure correct functionality.

The following commands may be executed (as root):

```
chown secure-user install_dir/nxnr/arch/acct

chgrp secure-group install_dir/nxnr/arch/acct
```

```
chmod ug+s install_dir/nxnr/*/acct

chmod o= install_dir/acct

chmod o= install_dir/acct/*
```

where *secure-user* is the userid that will own the files and *secure-group* is the groupid of the group that will own the files.

3.6 Determining System Limits

System resources can have a profound impact on the type and size of analyses that can be performed with NX Nastran. Resources that are too low can result in excessive time to complete a job or even cause a fatal error. For LINUX, the current resource limits on the local computer are obtained with the following command:

```
nxnr nastran limits
```

Note

- The limits can vary among users and computers. If a queuing system such as NQS or NQE is installed, different limits may also be found on the various queues.
- For LINUX, the stacksize is used to set the maximum length of the Nastran command line. If the stacksize is not set, then the maximum length is 1024 characters. If the stacksize is set, then the maximum length is the value of the stacksize or 128K, whichever is less.
- The output from the limits special function may specify “unlimited”. In this context, “unlimited” means there is no limit on your use of a resource that is less than those architectural limits imposed by the processor or the operating system.

  A more important interpretation of unlimited occurs when describing file size limitations. Table 5-7 lists those systems that support large files, i.e., in excess of 2 gigabytes. In this case, unlimited can mean 2**32-1 (4 294 967 295) bytes if large files are not supported, or upwards of 2**64-1 (18 446 744 073 709 551 615) bytes if large files are supported.

Sample output from this command for some of the various computers used to port NX Nastran follows.

X86–64 Opteron/Intel EM64T

```
Current resource limits:
CPU time: unlimited
Virtual address space: unlimited
Working set size: unlimited
Data segment size: unlimited
Stack size: 8192 KB
Number of open files: 1024 (hard limit: 1024)
File size: unlimited
Core dump file size: 0 MB
```

3.7 Customizing the Command Initialization File

The command initialization file, *install_dir*/bin/nast*r*.ini, is used to define keywords that are to be set whenever the nastran command is executed. Typical keywords defined in this file include the installation base directory and the version of NX Nastran.

Setting Command Initialization File Keywords

The following table lists the keywords that are generally set in the command initialization file.

Table 3-2. Command Initialization File Keywords

Keyword	Purpose
acct	Enables job accounting, see "Enabling Account ID and Accounting Data" .
acvalid	Activates account ID validation, see "Enabling Account ID and Accounting Data".
NXN_BASE	Defines the installation base directory. Normally this is defined as an environment variable by the nxn*r* command.
version	Specifies the default version of NX Nastran to run.

3.8 Customizing the Runtime Configuration Files

NX Nastran keywords ("Keywords") and NASTRAN statements, found in the section titled "Using the NASTRAN Statement", can be placed in runtime configuration (RC) files to set default or system-wide values.

NX Nastran uses the following RC files:

- System RC file

  This file is used to define parameters that are applied to all NX Nastran jobs using this installation structure.

  install_dir\conf\nastr.rcf

- Architecture RC file

  This file is used to define parameters that are applied to all NX Nastran jobs using this architecture.

  install_dir\conf*arch*\nastr.rcf

- Node RC file

  This file is used to define parameters that are applied to all NX Nastran jobs running on this node.

  install_dir\conf\net*nodename*\nastr.rcf

- User RC file

  This file is used to define parameters that are applied to all NX Nastran jobs run by an individual user.

  %HOMEDRIVE%%HOMEPATH%\nast*r*.rcf

- Local RC file

This file should be used to define parameters that are applied to all NX Nastran jobs that reside in the input data file's directory. This file is in the same directory as the input data file. If the "rcf" keyword is used, this local RC file is ignored.

nast*r*.rcf

Please note that the shorthand "~", to refer to your or another user's home directory, cannot be used in an RC file. In addition, environment variables are only recognized within the context of a logical symbol definition.

The order of precedence for duplicated entries is as follows (with number 1 representing the highest precedence):

1. NASTRAN statements in the input file.
2. Keywords on the command line.
3. Local RC file.
4. User RC file.
5. Node RC file.
6. Architecture RC file.
7. System RC file.

An example of an RC file follows.

```
NASTRAN SYSTEM(20)=0
NASTRAN BUFFSIZE=16385
mem=3m
```

```
ALWAYS PRINT BEGIN,END
$ CHANGE DEFAULT BUFFSIZE
$ run with 3 145 728 words
```

Setting RC File Keywords

Most of the command line keywords can be set in any of the RC files. Table 3-3 lists keywords that are generally set in the system, architecture, or node RC files:

Table 3-3. RC File Keywords

Keyword	Preferred RC file	Purpose
accmd	System	Command line to invoke accounting logger program.
acct	System	Enables job accounting.
acvalid	System	Enables account ID (acid) validation.
authorize	System	Specifies the licensing method.
lock	Any	Prevent further changes to a keyword's value.
memory	Node	Specifies a default memory allocation.
memorymaximum	Node	Specifies a maximum "memory" request.
ncmd	Architecture	Specifies the notify command when "notify=yes" is set.
news	System	Controls the display of the news file at the beginning of the .f06.

post	Architecture	Specifies commands to be run after each job is completed.
ppcdelta	Architecture	Specifies the value that is subtracted from the “CPU” keyword value to determine the NQS per-process CPU time limit.
ppmdelta	Architecture	Specifies the value that is added to the “memory” keyword value to determine the NQS per-process memory limit.
pre	Architecture	Specifies commands to be run before each jobs begins.
prmdelta	Architecture	Specifies the value that is added to the “ppm” value to determine the NQS per-request (per-job) memory limit.
qoption	Architecture	Specifies a string of additional queuing options to be set in the queue submittal command.
real	Node	Specifies the “REAL” parameter to limit virtual memory usage.
scratch	Any	Specifies the default job status as scratch or permanent.
sdirectory	Node	Specifies a default scratch directory.
submit	Architecture	Defines queues and their associated submittal commands.
sys*x*	Architecture	Specifies system cells.

Parameter Specification

NX Nastran allows the specification of PARAM statements as follows:

1. Inside of the input file.
2. Parameter values can be specified in the nast*r*.rcf (WINDOWS).
3. Parameters can be assigned a user defined keyword. The new keyword can then be used to specify a value for the parameter on the command line or in the nastran resource file. The keywords can be defined in the “nastran.params” file in the architecture directory.

Specifying Parameters in the Resource (rc) file

Parameters can be specified using the following syntax in the rc file:

PARAM,name,value

For example:

PARAM,POST,-2

PARAM,AUTOSPC,NO

Using the nastran.params file

The parameter defining keywords are assigned in the nastran.params file located at:

Linux: install_dir/nxn*r*/arch_dir/nastran.params

Windows: install_dir\nxn*r*\i386\nastran.params

You can override this default file using the “0.params” keyword on the command line. For example, if you have a custom defined parameter file named “my_specs.param”, the command line submittal would be:

nastran input_file.dat 0.params=full_path_name/my_specs.param

where “full_path_name” is the fully qualified path name of the file “my_specs.param”.

The parameters can be assigned to keywords in the nastran.params file as follows:

keyword name : param name : param value

where

- **Keyword name** – the name to be assigned as a mnemonic for the parameter name. This should be a single word without embedded white space. Also, this name should not be the same as any of the internal keywords. The keyword name can be the same as the parameter name.
- **Parameter Name** – the actual name of the parameter, for example, “autospc”.
- **Parameter Value** – used for syntax checking when the keyword name is used in the command line. The parameter value is either a “number” (integer/real) or an acceptable value list. For example, {“yes”,”no”}.

Comments can be included in the nastran.params file by starting the line with “#”, “$” or “;” as the first character.

Below is an example nastran.params file:

```
#
# example nastran.params file
mypost : POST : number
nxautospc: AUTOSPC : {“yes”, “no”}
alpha1: alpha1 : number,number
```

In the nastran submittal, these keywords can be used to assign values to the parameters:

```
nastran input_file.dat mypost=-2 nxautospc=no alpha1=0.1,0.2
```

The order of processing the parameter values follows the same precedence rules as in the “rc” file. That is, parameters in the input file have the precedence over the command line specification which has precedence over the rc file specification.

3.9 Limiting “memory” Requests

The nastran command provides a “memorymaximum” keyword that permits you to specify a maximum memory request on a site-wide, per-architecture, or per-node basis. This value can be set to any legal memory size.

The default values are

```
memorymaximum=0.8*physical
```

on Linux, and

```
memorymaximum=1.2*physical
```

on Windows. If this limit is exceeded, the nastran command will issue a UWM and reduce the memory request.

Note

As installed, the computer's physical memory is only known on Windows. Other systems must specify a hard limit or specify the physical memory size via the "s.pmem" keyword.

You may leave the default limits in place, or specify any value or values appropriate to your site.

It may be advisable to lock this keyword to ensure the limit is not removed. This is accomplished with the RC file entry

```
lock=memorymaximum
```

Note

Be sure you specify this line after any specification of the "memorymaximum" keyword.

3.10 Customizing the News File

NX Nastran includes a news file (*install_dir*/nxn*r*/nast/news.txt on Linux and *install_dir*\nxn*r*\nast\news.txt on Windows) that briefly describes important new features of the release. You can also use news file to distribute information to users.

There are two ways the news file can be viewed. The most common way is by specifying "news=yes" or "news=auto" on the command line or in an RC file. This specification will cause the news file to be printed in the .f06 file just after the title page block. The other method is by using the news special function

```
nxnr nastran news
```

This will display the news file on the screen.

3.11 Customizing the Message Catalog

NX Nastran uses a message catalog for many messages displayed in the .f06 file. The standard message catalog source file on Linux is:

```
install_dir/nxnr/util/analysis.txt
```

The standard message catalog source file on Windows is:

```
install_dir\nxnr\util\analysis.txt
```

This file may be modified to meet the needs of a site or a user.

Once the changes have been made, a message catalog is generated using the command

```
nxnr msgcmp myfile
```

where "myfile.txt" is the message catalog source file. This command will generate a message catalog in the current directory with the name "myfile.msg". The message catalog is identified with the "msgcat" keyword, and can be tested using the command

```
nxnr nastran msgcat=myfile.msg other_nastran_keywords
```

Once the message catalog has been validated, it may be installed on Linux with the command:

```
cp myfile.msg install_dir/nxnr/arch/analysis.msg
```

or installed on Windows with the command:

```
copy myfile.msg install_dir\nxnr\arch\analysis.msg
```

where *install_dir* is the installation base directory and *arch* is the architecture of the system using the message catalog. You will need write permission to the architecture directory to do this.

Note

Message catalogs are computer-dependent. Table7-1 identifies the systems that are binary compatible; binary compatible systems can use the same message file.

3.12 Defining a Computer Model Name and CONFIG Number

If the nastran command cannot identify a computer, the following message will be written to the screen before the NX Nastran job begins:

```
*** SYSTEM WARNING MESSAGE (nastran.validate_local_keywords)
    s.config=0     (program default)
      Default CONFIG value.
    A config number for this computer could not be
    determined.  Defining this computer in the model file
    install_dir/conf/arch/model.dat, using rawid=rawid; or
    defining <config> in an RC file may correct this
    problem.
```

There are two possible resolutions to this warning message. The preferred solution is to create the file *install_dir*/conf/*arch*/model.dat on Linux or *install_dir*\conf*arch*\model.dat on Windows with the model name and configuration number of the computer. This file contains zero or more lines of the form:

```
model, proc, rawid, config
```

where

model	is the name of the computer model. This string should be enclosed in quote marks if it contains spaces or commas.
proc	is the file type of the alternate executable. This value is set to null to select the standard executable. The “system” special function reports this name.
rawid	is the “rawid” value reported in the above message text or by the “system” special function.
config	The CONFIG number used to select the timing constants. If this value is null, *rawid* is used as the CONFIG number.

Any values in this table will override the default values built into the nastran command.

An alternative solution to creating this file is to set the *config* keyword in the node RC file, see “Setting RC File Keywords”. Note, however, this will not set a model name.

3.13 Generating a Timing Block for a New Computer

NX Nastran uses timing constants to determine the fastest algorithm or “method” to perform certain numerically intensive operations. Timing constants are installed for a variety of computers. If constants are not installed for your particular computer, NX Nastran will select default timing constants and display the following warning message:

```
*** USER WARNING MESSAGE 6080 (TMALOC)
THE TIMING CONSTANTS DATA BLOCK TIMEBLK NOT FOUND ON THE DELIVERY DATABASE FOR:
MACHINE = 5 CONFIG = 56 OPERASYS = 3 OPERALEV = 7 SUBMODEL = 1
LOADING DEFAULT TIMING CONSTANTS DATA BLOCK FOR:
MACHINE = 5 CONFIG = 56 OPERASYS = 3 OPERALEV = 5 SUBMODEL = 1
MODULE TIMING ESTIMATES INACCURATE AND MAY CAUSE INEFFICIENT JOB EXECUTION
```

Ignoring the message may result in excessive runtimes. Proper timing constants for a specific computer may be generated and installed by running a job that measures the timing constants of the computer and stores them in the delivery database.

Use the following steps to add timing constants for your computer to the delivery database:

1. Determine the architecture name of your system by consulting Table 3-1 or executing the command

   ```
   nxnr nastran system
   ```

2. Change the working directory to the architecture directory of your computer.

   ```
   cd install_dir/nxnr/arch
   ```

   on Linux, or

   ```
   cd install_dir\nxnr\arch
   ```

   on Windows, where *arch* was determined in Step 1 above.

3. Copy the Structured Solution Sequence files to be modified by the gentim2 run with the commands:

   ```
   cp SSS.MASTERA gentim2.MASTERA
   cp SSS.MSCSOU gentim2.MSCSOU
   cp SSS.MSCOBJ gentim2.MSCOBJ
   ```

   on Linux, or

   ```
   copy SSS.MASTERA gentim2.MASTERA
   copy SSS.MSCSOU gentim2.MSCSOU
   copy SSS.MSCOBJ gentim2.MSCOBJ
   ```

   on Windows.

4. Issue the command

   ```
   nxnr nastran DELDIR:gentim2 old=yes scratch=no batch=no
   ```

   on Linux, or

   ```
   nxnr nastran DELDIR:gentim2 old=yes scratch=no
   ```

   on Windows.

   This command runs the job “DELDIR:gentim2.dat”, where “DELDIR” is a pre-defined logical symbol pointing to the directory containing the solution sequence source files. The value of the Bulk Data parameter “PARAM” is set to 7 by default, as shown in the partial listing of gentim2.dat below

```
NASTRAN MESH SYSTEM(124)=-1
PROJ LTC LOAD TIMING CONSTANTS
INIT MASTER,LOGICAL=(MASTERA(5000))
INIT SCRATCH(NOMEM)
TIME 2000
SOL GENTIMS
CEND
BEGIN BULK
PARAM,PARAM,7
.
.
.
```

In general, the larger the value of “PARAM”, the longer the gentim2 job runs and the more accurate the timing results. If gentim2 runs for more than one hour, you may choose to reduce the value of “PARAM”, this will shorten the elapsed time of the gentim2 job.

5. If there are no errors, replace the old DBsets with the new DBsets created by the gentim2 run. Do this with the following commands:

```
mv gentim2.MASTERA SSS.MASTERA
mv gentim2.MSCOBJ SSS.MSCOBJ
mv gentim2.MSCSOU SSS.MSCSOU
```

on Linux, or

```
copy gentim2.MASTERA SSS.MASTERA
copy gentim2.MSCOBJ SSS.MSCOBJ
copy gentim2.MSCSOU SSS.MSCSOU
```

on Windows.

3.14 Customizing Queue Commands

The nastran command runs an NX Nastran job by validating the command line and RC files, generating a “job script” that will run the NX Nastran executable, and running that script. When the “queue” keyword is specified, the corresponding “submit” keyword defines the command used to run the job script. The submit keyword , only specified in RC files, consists of a list of queue names followed by the command definition for the queues as shown below:

`submit=`*`queue_list=command_definition`*

or

`submit=`*`command_definition`*

When specified, the *queue_list* contains one or more “queue” names separated by commas. If a queue list is not supplied (as shown in the second example), the *command_definition* applies to all queues.

The *command_definition* of the “submit” keyword value defines the command used to run a job when a “queue” keyword is specified that matches a queue name in a submit keyword’s queue_list. The command_definition can contain keyword names enclosed in percent “%” signs that are replaced with the value of the keyword before the command is run.

Note

- When defining queue commands, it may be useful to build the job script but not actually execute it. Use the “-n” option, for example

  ```
  nxnr -n nastran myjob queue=myqueue
  ```

- The examples presented below are only intended to illustrate the “submit”, “qopt” and “queue” keywords. The examples may not work with your queuing software.

- You must use the Korn shell to run the script generated by the nastran command.

Consider the following example:

```
submit=small,medium,large=qsub -q %queue% -x -eo -s /bin/ksh %job%
```

In this example, the “qsub” command is used to run a job when “queue=small”, “queue=medium”, or “queue=large” is specified.

Any keyword used by the nastran command may be specified in the “submit” keyword’s command definition. The most common keywords used in the command definition are:

Keyword	Value
after	Value specified with the “after” keyword
cpulimit	Value specified with the “cpulimit” keyword.
job	Name of the job script file built by the nastran command.
log	Name of the LOG file.
ppc	Value of “ppc”, i.e, (%cpulimit% - %ppcdelta%).
ppm	Value of “ppm”, i.e., (%memory% + %ppmdelta%).
prm	Value of “prm”, i.e., (%ppm% + %prmdelta%).
qclass	This can be used to define an optional queue class in the command definition.
qoption	This can be used to define any option not directly represented by the other variables or not explicitly included in the command definition.
username	User name

Using the previous example, the command

```
nxnr nastran example queue=small
```

runs the job script using the command:

```
qsub -q small -x -eo -s /bin/ksh example.J12345
```

The %queue% keyword reference is replaced by the specified queue, and the %job% keyword reference is replaced by the name of the execution script.

Keyword references can also contain conditional text that is included only if the value of the keyword is not null, or matches (does not match) a regular expression. A complete description of the keyword reference syntax is described in “Keyword Reference Syntax”. To check for a nonnull value, use the form

```
%kwd:condtext%
```

where *kwd* is the name of the keyword and *condtext* is the conditional text to be included. If the value of the keyword is null, the keyword reference is removed from the command. If the value of the keyword is not null, the keyword reference is replaced with the contents of *condtext*. Within *condtext*, the value of the keyword is represented by an open-close brace pair "{}".

For example:

```
submit=s=qsub -q %queue% %after:-a {}% -x -s /bin/ksh %job%
```

In this example, the "aft" keyword is references with conditional text. Using this example, the command

```
nxnr nastran example queue=s after=10:00
```

runs the job script using the following qsub command:

```
qsub -q s -a 10:00 -x -s /bin/ksh example.J12345
```

Using the same "submit" keyword, the command

```
nxnr nastran example queue=s
```

runs the job script using the following command:

```
qsub -q s -x -s /bin/ksh example.J12345
```

In this case, the "after" keyword was not specified and the entire contents of the %after% keyword reference was removed from the qsub command line.

Special Queues

When the "queue" keyword is not specified, the following three special queues are used:

Keyword	Queue Name	Command Definition
after	-aft	| at %after%
batch=yes	-bg	%job%
batch=no	-fg	%job%

Note

- If the first character of the command is the pipe character, "|", the contents of job script will be piped into the command.
- The command for the "-bg" queue is always executed in the background; the "-fg" and "-aft" commands are always executed in the foreground.

Changing the command definitions of these queues (using the "submit" keyword) will change the way the nastran command runs a job under the "after" and "batch" keywords.

3.15 Customizing the Script Templates

The nastran command relies on script templates to construct the job script that is built for every NX Nastran job. Several templates are provided: "*install_dir*/bin/nast1.dmp" is used for DMP jobs, "*install_dir*/bin/nast1.lcl" is used for serial or SMP jobs run on the local system, and

"*install_dir*/bin/nast1.rmt" is used for serial or SMP jobs run on a remote system using the "node" keyword. These templates may be modified to suit your needs.

Note

When customizing the script templates, it may be useful to build the job script but not actually execute it. Use the "-n" option, e.g.,

```
nxnr -n nastran myjob
```

Keyword Reference Syntax

The script templates use the keyword reference syntax that was partially introduced in the previous section. Table 3-4 provides examples.

Table 3-4. Keyword Syntax

Syntax	Value	Side effects
%%	%	
%keyword%	Value of keyword.	
%keyword:condtext%	condtext	
%keyword=re%	Value of the parenthetic expression if specified in the re, otherwise the string matched by the re.	
%keyword=re:condtext%	condtext if re is matched.	
%keyword!re:condtext%	condtext if re is not matched.	
%keyword:%		Kill remainder of line if *keyword* has null value. In a case construct, the default case.
%keyword=re:%		Kill remainder of line if *re* does not match.
%keyword!re:%		Kill remainder of line if *re* does match.
%keyword?:%		Start of case construct. See "Using Regular Expressions".
%keyword>cmp:context%	condtext if keyword is > than cmp	
%keyword>cmp:condtext%	condtext if keyword is ≥ than cmp	
%keyword<cmp:condtext%	condtext if keyword is < than cmp	
%keyword<cmp:condtext%	condtext if keyword is ≤ than cmp	
%keyword>cmp:%		Kill remainder of line if keyword is not > than cmp
%keyword>cmp%		Kill remainder of line if keyword is not ≥than cmp
%keyword<cmp:%		Kill remainder of line if keyword is not < than cmp

%keyword<cmp:%		Kill remainder of line if keyword is not ≤ than cmp

Keyword Reference Examples

The keyword reference syntax is described using the following examples from *install_dir*/bin/nast1.lcl.

Unconditional Keyword Substitution

```
export NXN_BASE=%NXN_BASE%
```

The keyword reference %NXN_BASE% will be replaced by the value of the “NXN_BASE” keyword.

```
export DBSDIR=%dbs=\(.*\)/%
```

The keyword reference %dbs=\(.*\)/% will be replaced with the value of the parenthetic regular expression. For example, given the keyword value “onedir/anotherdir/myfile”, the parenthetic expression is “onedir/anotherdir”, and the substituted line would read:

```
export DBSDIR=onedir/anotherdir
```

Conditional Keyword Substitution

```
%sysfield:SYSFIELD={}%
```

The keyword reference %sysfield:SYSFIELD={}% will be replaced by the string “SYSFIELD=keyword-value” if and only if the keyword is not null.

```
%dcmd=dbx:run%
```

The keyword reference %dcmd=dbx:run% will be replaced by “run” if and only if “dcmd=dbx” was specified. If the equal sign in the keyword reference was replaced by an exclamation mark, i.e., %dcmd!dbx:run%, then the keyword reference will be replaced by “run” if and only if “dcmd” was set to a nonnull value not equal to “dbx”.

Conditional Inclusion

```
%NXN_ARCH=x86_64linux:%startdate=date +%%a %%h %%d %%H:%%M:%%S %%Z %%Y
%NXN_ARCH!x86_64linux:%startdate=date
```

Conditional inclusion is indicated by a null conditional text string; i.e., the colon is immediately followed by a percent sign. This capability is generally used with a regular expression to include the remainder of the line if a keyword value matches or does not match a regular expression. In the first line, the remainder of the line will be included if the “NXN_ARCH” keyword contains the string “x86_64linux” while the remainder of the second line will be included if “NXN_ARCH” does not contain the string “x86_64linux”. More than one conditional inclusion keyword reference can be used on a line to create more complex tests.

```
%prt=y:%%pdel=y:%/bin/rm %out%.f04 %out%.f06 %out%.log
```

The “rm” command will included if and only if “prt=yes” and “pdel=yes”.

A “case” structure is specified as follows:

```
...%s.model?:%
...%s.model=IP.$:% SGI_ISA=mips1; export SGI_ISA
...%s.model=IP12:% SGI_ISA=mips1; export SGI_ISA
...%s.model=IP15:%  SGI_ISA=mips1; export SGI_ISA
...%s.model=:%      SGI_ISA=mips2; export SGI_ISA
```

This sequence will result in the line

```
SGI_ISA=mips1
```

if “s.model” is “IP” followed by a single character (using the second line), or “IP12" (using the third line), or “IP15" (using the fourth line), otherwise

```
SGI_ISA=mips2
```

will be generated using the last line. Case constructs can be nested, but a keyword may only be active in one case at a time.

Greater and less-than comparisons can be used instead of regular expression matching to control conditional inclusion. These comparisons are done with integer, floating, or string values based on the types of the two values.

```
%a.release>68: %CONFIG=%config%
```

The CONFIG statement will be included if “a.release” is greater than 68.

Nested Keyword Values

One level of nested keywords may occur anywhere within the %.*% string. Only unconditional keywords substitutions are supported for nested keywords. Nested keywords are specified as \%keyword\%.

```
%dmparallel>\%maxnode\%:%#@ node = %maxnode%
```

This sequence will cause the “#@ node ..” text to be included if the value of the “dmparallel” keyword is greater than the value of the “maxnode” keyword.

Chapter 4: Installation and Configuration of DMP

The information in this chapter has been moved into Chapter 7 of the NX Nastran Parallel Processing Guide. You can open this guide from the NX Nastran Help Library.

Chapter 5: Using the Basic Functions of NX Nastran

This chapter is directed to the engineer running NX Nastran. It covers using the nastran command, including file types, filenames, logical symbols, the help facility, and other functions. In addition, this chapter provides an overview of the basic keywords, outlines resource requirements, describes how to specify memory sizes, introduces the sample problem libraries, and how to make file assignments, as well as how to use databases, how to apply the INCLUDE statement, and how to resolve abnormal terminations.

5.1 Running NX Nastran

NX Nastran jobs are generally run using the nastran command, which is located at *install-directory*/bin. The basic format of this command is

```
nxnr nastran input_file keywords
```

where *input_file* is the name of the file containing the input data and *keywords* is zero or more optional keyword assignments. For example, to run an NX Nastran job using the data file example.dat, enter the following command:

```
nxnr nastran example
```

The nastran command is one way to run the software. The various commands in *install-directory*/bin are described in LP-64 and ILP-64 Executable Information.

Various options to the nastran command are available using keywords described in "Keywords". Keyword assignments consist of a keyword, followed by an equal sign, followed by the keyword value, for example:

```
nxnr nastran example scratch=yes
```

Note

In Windows you can use a hash mark "#" instead of the equal sign. This is useful if the nastran command is being placed in a ".bat" file.

```
nxnr nastran example scratch#yes
```

Keyword assignments can be specified on the command line or included in RC files.

There are two RC files controlled by you:

- The user RC file is used to define parameters applicable to all NX Nastran jobs you run.

  %HOMEDRIVE%%HOMEPATH%\nast*r*.rcf

- The local RC file should be used to define parameters applicable to all NX Nastran jobs that reside in the input data file's directory, and is located in the same directory as the input data file. If the "rcf" keyword is used, this local RC file is ignored.

  nast*r*.rcf

- In general, spaces and special characters can be included in your path and file names if you include single or double quotes around the string. Although, you should avoid spaces if are defining your pathname using an environment variable. In addition, the single quote should not be embeded in your path or file name. For example, this path results in a fatal error:

  ```
  "D:\inputfile'\glue101n.dat"
  ```

Using Regular Expressions

The regular expression syntax supported by the nastran command is compatible with the standard ed(1) regular expression syntax with the exception that only one parenthetic expression is permitted.

One-character Regular Expressions

- Any character, except for the special characters listed below, is a one-character regular expression that matches itself.
- A backslash, “\”, followed by any special character is a one-character regular expression that matches the special character itself. The special characters are: period, “.”, asterisk, “*”, and backslash “\”, which are always special except when they appear within brackets; circumflex, “^”, which is special at the beginning of a regular expression or when it immediately follows the left bracket of a bracketed expression; and dollar sign “$”, which is special at the end of a regular expression.
- A period, “.”, is a one-character regular expression that matches any character.
- A nonempty string of characters enclosed within brackets, “[” and “]”, is a one-character regular expression that matches one character in that string. If, however, the first character of the string is a circumflex, “^”, the one-character regular expression matches any character except the characters in the string. The circumflex has this special meaning only if it occurs first in the string. The dash, “-”, may be used to indicate a range of consecutive characters. The dash loses this special meaning if it occurs first (after an initial circumflex, if any) or last in the string. The right square bracket, “]”, does not terminate such a string when it is the first character within it (after an initial circumflex, if any).

Regular Expressions

- A one-character regular expression is a regular expression that matches whatever the one-character regular expression matches.
- A one-character regular expression followed by an asterisk, “*”, is a regular expression that matches zero or more occurrences of the one-character regular expression. If there is any choice, the longest leftmost string that permits a match is chosen.
- A one-character regular expression followed by “\{*m*\}”, “\{*m*,\}”, or “\{*m*,*n*\}” is a regular expression that matches a ranges of occurrences of the one-character regular expression. The values of *m* and *n* must satisfy $0 \le m \le n \le 254$; “\{*m*\}” exactly matches *m* occurrences; “\{*m*,\}” matches at least *m* occurrences; “\{*m*,*n*\}” matches any number of occurrences between *m* and *n* inclusive.
- A concatenation of regular expressions is a regular expression that matches the concatenation of the strings matched by each component of the regular expression.
- A regular expression enclosed between the character sequences “\(” and “\)” defines a parenthetic expression that matches whatever the unadorned regular expression matches. Only one parenthetic expression may be specified.
- The expression “\1" matches the same string of characters as was matched by the parenthetic expression earlier in the regular expression.

Constraining Regular Expressions

- A circumflex, “^”, at the beginning of an entire regular expression constrains the regular expression to match an initial segment of a string.
- A dollar sign, “$”, at the end of an entire regular expression constrains the regular expression to match a final segment of a string.
- The construction “^*re*$” constrains the regular expression to match the entire string.
- The construction “^$” matches a null string.

File Types and Versioning

NX Nastran’s default input and output files use the following types:

Type	Type of File	Description of File
.dat	Input	Input Data File
.f04	Output	Execution Summary File
.f06	Output	Output Data File
.log	Output	Job Log File
.op2	Input Output	OUTPUT2 File
.pch	Output	Punch File
.plt	Output	Binary Plot File
.xdb	Output	Results Database

Note

1. If the input file is specified as “example” and the files “example.dat” and “example” both exist, the file “example.dat” will be chosen. In fact, it is impossible to use a file named “example” as the input data file if a file named “example.dat” exists.
2. The “jidtype” keyword may be used to specify an alternate default suffix for the input data file. For example, “jidtype=bdf” will change the default file type to “.bdf”.
3. The XDB file is not versioned.
4. The “oldtypes” keyword may be used to specify a list of additional file types that are versioned. For example, “oldtypes=xdb” will cause the XDB file to be versioned.

When a job is run more than once from the same directory, the previous output files are versioned, or given indices. The indices are integers appended to the filename; the same integer will designate files for the same job. For example:

v2401.f04	v2401.f04.1	v2401.f04.2	v2401.f04.3
v2401.f06	v2401.f06.1	v2401.f06.2	v2401.f06.3

The files listed (according to time of execution from oldest to newest) are:

v2401.f04.1
v2401.f04.2
v2401.f04.3
v2401.f04

v2401.f06.1
v2401.f06.2
v2401.f06.3
v2401.f06

Using Filenames and Logical Symbols

Several of the parameters used by NX Nastran, including command line arguments, initialization and RC file commands, and statements within NX Nastran input files, specify filenames. The filenames must follow your system's standard filename conventions, with the addition that filenames can include a "logical symbol" component, i.e., the filename can be specified in either of the following forms:

```
filename
logical-symbol:filename
```

Logical symbols provide you with a way of specifying file locations with a convenient shorthand. This feature also allows input files containing filename specifications to be moved between computers without requiring modifications to the input files. Only the logical symbol definitions that specify actual file locations need to be modified.

Only one logical symbol name may be used in a filename specification. This logical symbol must be the initial component of the filename string, and it must be separated from the filename by a colon ":". If the symbol has a non-null value, the actual filename is created by replacing the symbol name with its value and replacing the colon with a slash; otherwise, both the symbol name and the colon are left as is.

Note

- A logical symbol can be defined using any environment variable or previously defined symbol. Use the standard environment variable reference convention, i.e., "$*name*" or "${*name*}" on Linux and "%*name*%" on Windows.
- When using the "hosts=" or "node=" keywords, the symbol keyword defines symbolic names on the remote systems. Alternately, the lsymbol keyword can be used to define symbolic names for the local system. See "symbol" and "lsymbol" for more information.
- Logical symbols must be more than one character long, i.e., the filename reference "D:\temp\myfile.dat" will be interpreted on Windows as a drive reference followed by a pathname.
- NX Nastran accepts Windows pathnames using the slash "/" character as a replacement for the backslash "\".

For example, assume that your home RC file contains the line

```
SYMBOL=DATADIR=/dbs/data
```

on Linux, or

```
SYMBOL=DATADIR=d:\dbs\data
```

on Windows, and a job is submitted with the command

```
nxnr DATADIR:nastran example
```

Since NX Nastran automatically sets the OUTDIR environment variable to the value of the “out” keyword, the filenames

```
'DATADIR:myfile.dat'
  'OUTDIR:testdata.info'
```

will reference the files

```
/dbs/data/myfile.dat
./testdata.info
```

on Linux and

```
d:\dbs\data\myfile.dat
.\testdata.info
```

on Windows respectively, see “symbol” for more information.

Several other symbols are automatically created by the nastran command. These include DELDIR, DEMODIR, TPLDIR, and SSSALTERDIR to access the delivery database source directory, and DEMO, TPL, and SSSALTER libraries, respectively.

Using the Help Facility and Other Special Functions

Several special functions are supported by reserved input data filenames. If these names are specified as the input data file, the nastran command will execute the special function and exit.

Note

If you need to use one of these reserved names as an actual input filename, you must either prefix the filename with a path or append a file type to the filename.

The special functions are invoked as follows:

```
nxnr nastran help
```

This request will display the basic help output. Additional help capabilities are described in the basic help output.

```
nxnr nastran help keyword1 [keyword2 ...]
```

This request will display help for the keywords listed on the command line.

```
nxnr nastran limits
```

This request will display the current resource limits.

```
nxnr nastran news
```

This request will display the news file.

```
nxnr nastran system
```

This request will display system information about the current computer.

These requests can be executed on a remote computer that has NX Nastran installed by also specifying the keyword “node=*nodename*”, for example:

```
nxnr nastran system node=thatnode
```

5.2 LP-64 and ILP-64 Executable Information

With finite element model sizes becoming larger, the need for increased memory allocation has become more important. In response to this need, NX Nastran executables are compiled with a 64-bit integer size. The 32-bit integer executable can allocate up to 8 Gb of memory, while the executable compiled with a 64-bit integer size can allocate approximately 20 million terabytes. Practically speaking, there are no machines currently supporting more than half a terabyte, thus the amount of memory these executables can allocate is only limited by the amount of memory installed on the machine.

There are 2 different executable types available for NX Nastran:

- 32-bit word size and 64-bit memory pointer size, designated LP-64. Integers are 32-bits and floating point uses two 32-bit words.

- 64-bit word size and 64-bit memory pointer size, designated ILP-64. Integers are 64-bits and floating point uses one 64-bit word.

When the LP-64 executable is used, the bytes_per_word is 4. When the ILP-64 executable is used, the bytes_per_word is 8. This difference is important when you are specifying memory with the “memory” keyword. See the “memory” keyword in the NX Nastran Quick Reference Guide for more information.

Understanding the commands in /bin

The commands in *install_location*/bin depend on the system and installation. The commands provided for each installation are described in the table below.

Windows Install

Command	Description
nastranw.exe nastran64w.exe	Runs a job using the LP-64 executable. When executed, they will bring up the dialogs to select the input file and enter keywords. A Windows shortcut is created for nastranw.exe during the installation.
nastran.exe nastran64.exe	Runs a job using the LP-64 executable. If executed directly, it will bring up the dialogs to select the input file and enter keywords. The input format from a DOS or shell prompt is `nastran input_data_file.dat keywords`
nastr.exe*	Runs a version specific job using the LP-64 executable. The input format from a DOS or shell prompt is `nastr input_data_file.dat keywords`
nxnr.exe*	Runs version specific utilities using the LP-64 executable. The input format from a DOS or shell prompt is `nxnr utility_name input_file keywords`
nastran64Lw.exe	Runs a job using the ILP-64 executable. It should be executed directly, and will bring up the dialogs to select the input file and enter keywords. You can manually use to create a Windows shortcut.

nastran64L.exe	Runs a job using the ILP-64 executable. If executed directly, it will bring up the dialogs to select the input file and enter keywords. The input format from a DOS or shell prompt is `nastran64L input_data_file.dat keywords`
nxnrL.exe*	Runs version specific utilities using the ILP-64 executable. The input format from a DOS or shell prompt is `nxnrL utility_name input_file keywords`

X86_64 Linux

Command	Description
nastran	Runs a job using the LP-64 executable. The input format is `nastran input_data_file.dat keywords`
nastr*	Runs a version specific job using the LP-64 executable. The input format is `nastr input_data_file.dat keywords`
nxnr*	Runs version specific utilities using the LP-64 executable. The input format is `nxnr utility_name input_file keywords`
nastran64L	Runs a job using the ILP-64 executable. The input format is `nastran64L input_data_file.dat keywords`
nxnrL*	Runs version specific utilities using the ILP-64 executable. The input format is `nxnrL utility_name input_file keywords`

* Note: The version specific commands like nastr and nxnr allow you to install multiple NX Nastran versions in the same hierarchy. For example, if you install versions 7 and 8 together, /bin would contain a nast8, nast7, nxn8, nxn7, and a single nastran.exe. Then the nastran command will run a typical job using the version last installed. The nast8 and nast7 commands will run a typical job, but for a specific version. The nxn8 and nxn7 commands will run version specific utilities.

ILP-64 executable file formats

The LP-64 executable writes binary output files as 32-bit. However, the ILP-64 produces a different binary file format since all integers and floating point data are written out with a 64-bit precision. Depending on the use of the binary output files from a 64-bit machine, you may need to convert a 64-bit files's format back to 32-bit. For example, post-processors currently only support 32-bit integers, thus the need to convert .op2 files to 32-bit. Three system cells are available to convert binary output files from 64- bit machines to 32-bit:

- Nastran OP2FMT=1 (or set nastran system cell 413=1): converts a 64-bit integer .op2 file to 32-bit integer format on an ILP-64 machine.
- Nastran OP4FMT=1 (or set nastran system cell 415=1): converts a 64-bit integer .op4 file to 32-bit integer format on an ILP-64 machine.
- Nastran INP4FMT=1 (or set nastran system cell 416=1): allows a 32-bit integer .op4 file to be read with the INPUT4 module on an ILP-64 machine.

The NASTRAN statements listed will override the OP2FMT, OP4FMT and INP4FMT parameters.

Note

Including PARAM,POST,n where “n”=-1 or -2, NX Nastran will automatically convert the 64–bit integer op2 file to a 32–bit op2 file. To override this, you will need to include PARAM,OP2FMT,64 in the bulk data section.

In addition to binary file format changes, the .f04 and .f06 output files will have the following differences when written from ILP-64 machines:

- The matrix trailers and the format of floating point numbers will change since all matrices that were double-precision will now show as single-precision.
- The exponent descriptor will be an “E” instead of a “D”.

ILP-64 Limitations

The ILP-64 executables have the following limitations:

- You only convert those data blocks that are NDDL defined from 64-bit to 32-bit. See chapter 3 of the NX Nastran DMAP Programmer’s Guide for more information on NDDL.
- All .op2 files written during a solution are in one precision format, that is, they are all either 32-bit or 64-bit precision .op2 files.
- The INPUT2 files are not converted.

5.3 Using the Basic Keywords

The following tables provide a partial list of the basic keywords that may be used on the command line or placed into RC files as appropriate. More advanced keywords are listed in “Using the Advanced Keywords”, and a complete list of all keywords and their syntax is listed in “Keywords”.

All Systems

Keyword	Purpose
append	Combines the .f06, .f04, and .log files into a single file after the jobs completes.
dbs	Specifies an alternate name for user database files.
memory	Specifies the amount of memory to be used by the job.
old	Renames existing output files with version numbers or deletes existing output files.
out	Specifies an alternate name for output files.
rcf	Specifies an alternate name of the local RC file.
scratch	Indicates databases are to be deleted when job completes.
sdirectory	Specifies an alternate scratch file directory.
symbol	Defines a symbolic name and value.

Linux Systems

Keyword	Purpose
after	Holds the job until the specified time.
batch	Runs the job in background or foreground.

Queuing (Linux)

Note

These capabilities depend upon the queue submission commands defined by the “submit” keyword and your queuing system. The keywords may not work on your system.

Keyword	Purpose
cpulimit	Specifies maximum CPU time to be allowed.
queue	Specifies name of queue where the job will be submitted to.

5.4 Specifying Memory Sizes

Several nastran keywords specify memory sizes. In all cases, the value can be specified either as the number of words (32-bit or 64–bit words) or as a number followed by one of the following modifiers:

Table 5-1. Memory Size Specifications

Specification	Size (Words)
nG, nGW	$n(1024^3)$
nGB	$n\frac{(1024^3)}{bpw}$
nM, nMW	$n(1024^2)$
nMB	$n\frac{(1024^2)}{bpw}$
nK, nKW	$n(1024)$
nKB	$n\frac{1024}{bpw}$
n, nW	n
nB	$\frac{n}{bpw}$

*n**physical, *n*xphysical	$n \cdot memory_{physical}$
*n**virtual, *n*xvirtual	$n \cdot memory_{virtual}$

where *bytes_per_word (bpw)* is 4 using the LP-64 executable, and 8 using the ILP-64 executable; “physical” is the computer’s physical memory, i.e., the “RAM”; and “virtual” is the swap size on Linux systems, and the maximum paging file size on Windows systems.

Examples are

```
nxnr nastran memory=1gb
```

Set the memory request to one gigabyte, 1024 megabytes, 1048576 kilobytes, 1073741824 bytes, or 268435436 words.

```
nxnr nastran memory=0.5xPhys
```

Set the memory request to 50% of the computer’s physical memory.

Maximum Memory Size

Table 5-3 lists the maximum “memory” size for NX Nastran platforms. A “memory” request larger than this value results in an error as the job starts.

Note

The actual maximum value you can specify depends on several factors, including the swap file size on Linux systems, the paging file size on Windows systems, and your virtual memory limit on most Linux systems. You must also deduct from the maximum value the size of the executable and the space required for the various operating system and Fortran runtime libraries.

Table 5-2. Maximum Memory Size (LP-64 platforms)

Platform	Memory
X86_64 Linux	7.99GB
Windows	7.99GB

Table 5-3. Maximum Memory Size (ILP-64 platforms)

Platform	Memory
X86_64 Linux	2,000,000 TB
Windows	2,000,000 TB

5.5 Determining Resource Requirements

For most models of moderate size (up to 5000 grid points for static analysis), you need not be concerned with resource requirements since the default NX Nastran parameters allocate sufficient resources. The analysis of larger models may require you to check the resource requirements and the various options that are available to manage memory and disk resources.

Detailed resource estimates can be obtained from the ESTIMATE program, described in "ESTIMATE". ESTIMATE reads the input data file and calculates the job's memory and disk requirements. The ESTIMATE program is most accurate in predicting the requirements of static analyses that don't have excessive output requests. The memory requirements for normal modes analyses using the Lanczos Method are reasonably accurate; however, the disk requirements are dependent upon the number of modes. This is a value that ESTIMATE does not know. Memory and disk requirements for other solutions are less accurate.

The best estimates of the memory requirements for a job are available in User Information Message 4157, but this requires an NX Nastran run.

Estimating BUFFSIZE

Table 5-4 presents recommendations for BUFFSIZE based on model size. These values have been chosen to represent the best compromise between database access speed and storage requirements for typical problems. An excessively large BUFFSIZE can result in more I/O data transferred and wasted space in the database for smaller problems; an excessively small BUFFSIZE can result in increases I/O counts for larger problems. You may be able to achieve higher performance or smaller databases using other values.

Table 5-4. Suggested BUFFSIZE Values

Degrees of Freedom	BUFFSIZE
DOF ≤100000	8193
100000 < *DOF* ≤ 200000	16385
DOF > 400000	32769 or 65537

Note

The actual I/O transfer size is (*BUFFSIZE* – 1) x *bpw* where *bpw* is 4 using the LP-64 executable, and 8 using ILP-64 executables.

5.6 Using the Test Problem Libraries

Three libraries of test problems are delivered with NX Nastran.

- The demonstration problem library (DEMO) contains a selection of NX Nastran input files. These files are accessible via the DEMODIR symbol, or via the path *install_dir*/nxn*r*/nast/demo on Linux and *install_dir*\nxn*r*\nast\demo on Windows.

- The test problem library (TPL) contains a general selection of NX Nastran input files showing examples of most of the NX Nastran capabilities. In general, these files are not documented. The files are accessible via the TPLDIR symbol, or via the path *install_dir*/nxn*r*/nast/tpl on Linux, and *install_dir*\nxn*r*\nast\tpl on Windows.

The DEMO and TPL libraries contain “demoidx.dat” and “tplidx.dat” respectively. These files contain one-line descriptions of the library members. Also included are files named “tplexec” and “demoexec”, which are scripts used to run the problems on Linux, or “tplexec.bat” and “demoexec.bat”, which are batch files used to run the problems on Windows.

If you only want to run a job from the DEMO or TPL libraries, the easiest method is to use either the “DEMODIR” or “TPLDIR” symbols, running the command from any convenient directory. For example,

```
nxnr nastran DEMODIR:d10101d
```

If you want to experiment with the file, copy the file to your own directory and then execute the problem. Note that several of the library files have “INCLUDE” files that should also be copied if they too will be modified, or they can be referenced as-is via the standard INCLUDE file processing; see “Using the INCLUDE Statement”.

Some example problems contain references to files that are qualified with the following logical symbols:

TPLDIR

DEMODIR

DBSDIR

OUTDIR

Unless they already exist in your environment as environment variables, the logical symbols DEMODIR and TPLDIR automatically point to the DEMO and TPL libraries respectively. DBSDIR and OUTDIR are always based on the “dbs” and “out” keywords respectively.

5.7 Making File Assignments

Using the ASSIGN statement, you can assign physical files used by NX Nastran to FORTRAN units or DBset files. The ASSIGN statement is documented in the File Management Section (FMS) of the *NX Nastran Quick Reference Guide*.

ASSIGN Statement for FORTRAN Files

For FORTRAN files, the format of the ASSIGN statement is

```
ASSIGN logical-name=filename, [ STATUS={NEW|OLD|UNKNOWN} UNIT=u,
FORM={FORMATTED|UNFORMATTED} TEMP DELETE SYS=sys-spec ]
```

There are no values of the SYS field defined for FORTRAN files on any system.

Table 5-5. FORTRAN Files and Their Default Attributes

Logical Name	Physical Name	Unit No.	Form	Status	Assignable	Open	Access	Description
SEMTRN	*sdir*/*data*.f01	1	FORMATTED	NEW	NO	YES	SEQ.	Input Data Copy Unit

LNKSWH	*sdir*/*data*.f02	2	UNFORMATTED	NEW	NO	YES	SEQ.	Link Switch Unit
MESHFL	*sdir*/*data*.f03	3	FORMATTED	NEW	NO	YES	SEQ.	Input Data Copy Unit
LOGFL	*out*.f04	4	FORMATTED	NEW	NO	YES	SEQ.	Execution Summary Unit
INPUT	*data*.dat	5	FORMATTED	OLD	NO	YES	SEQ.	Input File Unit
PRINT	*out*.f06	6	FORMATTED	NEW	NO	YES	SEQ.	Main Print Output Unit
PUNCH	*out*.pch	7	FORMATTED	NEW	YES	YES	SEQ.	Default Punch Out put Unit
INCLD1								Unavailable for Use
CNTFL								Unavailable for Use
INPUTT2	REQ	REQ	UNFORMATTED*	OLD	YES	NO	SEQ.	INPUTT2 Unit
OUTPUT2	*out*.op2	12	UNFORMATTED*	NEW	YES	YES	SEQ.	OUTPUT2 Unit
INPUTT4	REQ	REQ	UNFORMATTED	OLD	YES	NO	SEQ.	INPUTT4 Unit
OUTPUT4	REQ	REQ	UNFORMATTED†	NEW	YES	NO	SEQ.	OUTPUT4 Unit
PLOT	*out*.plt	14	UNFORMATTED	NEW	YES	YES	SEQ.	Plotter Output Unit
OUTPUT2	advnlin.op2	21, 22, 23	UNFORMATTED	NEW	NO	NO	SEQ.	Intermediate file
DBMIG								Unavailable for Use
DBC	*out*.xdb	40	UNFORMATTED	NEW	YES	YES	DIRECT	Database Converter Unit
DBUNLOAD	REQ	50	UNFORMATTED*	NEW	YES	NO	SEQ.	Database Unload
DBLOAD	REQ	51	UNFORMATTED*	OLD	YES	NO	SEQ.	Database Load
USER FILE	REQ	REQ	REQ	REQ	YES	NO	SEQ.	Any User-Defined File
OUTPUT2	advnlin.op2	75, 76	UNFORMATTED	NEW	NO	NO	SEQ.	Intermediate file

where:

Logical Name — The logical name used by NX Nastran.
The default name used to open the file.

Physical Name — “REQ” means that this parameter is required in the ASSIGN statement from the user.
The default FORTRAN unit number used by NX Nastran.

Unit Number (#) — “REQ” means that this parameter is required in the ASSIGN statement from the user.

Form	The default form used when the file is opened.
Status	The default status used when the file is opened.
Assignable	If “YES”, the user may assign a physical file to this logical name. If “NO”, the unit and logical names are reserved by NX Nastran.
Open	If “YES”, the file is opened by default. If “NO”, the file must be explicitly opened.
Access	If “SEQ.”, the file is opened for sequential access. If “DIRECT”, the file is opened for direct access.
*	FORMATTED is required for neutral-format files.
†	This must be FORMATTED if the BCD option is selected in DMAP.

ASSIGN Statement for DBsets

```
ASSIGN logical-name=filename [ TEMP DELETE SYS=sys-spec ]
```

See “Using the SYS Field” for details on the SYS field for DBsets.

Scratch DB Set Names

The default base name for scratch DB Sets now uses the base name of the input data file as a prefix; this will permit you to more easily identify the job that created specific files in the scratch directory.

Example: Linux: `nxnr nastran example sdir=/tmp`
Windows: `nxnr nastran example sdir=c:\temp`

The SCRATCH DBSet names will be named “/tmp/example.T<unique>.*” on the Linux systems and “c:\temp\example.T<unique>.*” on Windows systems where “<unique>” is a string created from the process ID of the nastran command and the current time.

Table 5-6. Default DBsets and Their Default Attribute

DBset	Memory			BUFFSIZE		Physical File Attribute		
	Type	Size	Units	Assignable	Size	Logical Name	Physical Name	SIZE
MASTER	RAM	120000	Words	YES	8193	MASTER	dbs.MASTER	5000
DBALL	N/A	-		YES	8193	DBALL	*dbs*.DBALL	1000000
OBJSCR	N/A	-		NO	8193	OBJSCR	*sdir*.OBJSCR	5000
SCRATCH	SMEM	100	Various	YES	8193	SCRATCH	*sdir*.SCRATCH	1000000
SCRATCH	N/A	-		YES	8193	SCR300	*sdir*.SCR300	1000000
User DBset	N/A	-		YES	8193	DBset	*dbs.DBset*	25000

where:

DBSet	The DBSet name.
Memory	The size of open core memory (in words) of the RAM of the MASTER DBset. The size may be modified using the FMS statement, INIT MASTER (RAM = value).
BUFFSIZE	The buffer size (words) used for I/O transfer for each DBset. This size may be changed if “YES” is in the Assignable column.
Logical Name	The logical name of the DBset. This name may be set with the ASSIGN or INIT statement.
Physical Name	The name of the file as known to your operating system. This name may be changed by using the ASSIGN statement.
Size	The default maximum file size (in GINO blocks) allowed for each DBset. This size may be changed by using the INIT statement.

5.8 Using Databases

NX Nastran provides a database for the storage and subsequent retrieval of matrices and tables. This facility consists of several database sets (DBsets) that conform to the following specifications:

- The NX Nastran limit on the maximum number of DBsets for an analysis is 200. Your computer may have a lower limit on the maximum number of open files that a process can open. This limit is displayed as the “Number of open files” by the “limits” special function. See “Using the Help Facility and Other Special Functions”.

- Each DBset may consist of 1 to 20 physical files. Again, this is subject to the maximum number of open files that your system permits.

- The maximum size of each DBset is machine dependent. There are several factors affecting the maximum size a given file can reach. Among these are: the job’s file resource limit; the available space of the file system containing the file; the maximum file size supported by the operating system, and the BUFFSIZE.

- On Linux systems, the “df” command lists the maximum space and available space in a file system. Your resource limit is displayed by as the “Maximum file size” by the “limits” special function.

- Systems now have 64-bit processors and now support “large files,” i.e., a file that can exceed 2 GB. NX Nastran supports large files on these systems. Table 5-7 summarizes.

Table 5-7. Database I/O Capabilities

Computer	Large File	File Mapping	Buffered I/O
Windows	Yes	Yes[4]	Yes
X86_64 Linux	Yes	Yes	Yes

Note

1. Large files are available if the file system containing the file supports large files. See your system administrator to determine which file systems, if any, support large files.

2. Large files can only be created on file systems supporting large files (the flags value from “df -g” must show the 0x10 bit set).

3. Large files are supported on ext2, ext3, xfs, jfs, and reiserfs filesystems. For other filesystems, please contact your Linux provider for information. Siemens PLM Software has tested large file support under kernels 2.4.x and 2.6.x on released systems.

4. File mapping is not recommended for Windows.

5. Large files can only be created on “XFS” file systems.

The default database provides for five DBsets that are subdivided into two categories (scratch and permanent DBsets) as follows:

- Three DBsets are scratch DBsets that are typically deleted at the end of a run. The logical names for these DBsets are SCRATCH, SCR300, and OBJSCR.

- The remaining two DBsets have the default names of dbs.MASTER and dbs.DBALL, where dbs is set by the “dbs” keyword.

The database may be defined in two different ways:

1. Using the “dbs” keyword on the command line; see “Using the “dbs” Keyword”.

2. Using ASSIGN statements in the FMS section of the input data file. See “ASSIGN Statement for DBsets” and “Using the ASSIGN Statement”.

Using the “dbs” Keyword

To illustrate the use of the “dbs” keyword, see the TPL file “am762d.dat”

```
ID UGS, AM762D $ JFC 30SEP88
$ DBS=AM762D SPECIFIED WHEN JOB SUBMITTED
TIME 2
SOL 101 $ SUPERELEMENT STATICS
CEND
TITLE = EXAMPLE: SPECIFY DBS=AM762D WHEN JOB SUBMITTED AM762D
SUBTITLE = COLD START
LOAD = 11
DISPLACEMENT = ALL
ELFORCE = ALL
BEGIN BULK
CBEAM,1,1,10,20,0.,1.,0.
FORCE,11,20,,100.,1.,.8,1.
GRID,10,,0.,0.,0.,,123456
GRID,20,,10.,0.,0.
MAT1,100,1.+7,,.3
PBEAM,1,100,1.,.08,.064,,.1
ENDDATA $ AM762D
```

To run this job, enter

```
nxnr nastran TPLDIR:am762d
```

The default value for “dbs” in this example is “./am762d” on Linux and “.\am762d” on Windows. The DBALL and MASTER DBsets are created in your directory as “am762d.DBALL” and “am762d.MASTER” respectively; and the output files are “am762d.f04”, “am762d.f06”, and “am762d.log”.

To restart from the previously created DBsets, use the following command:

```
nxnr nastran TPLDIR:am762r dbs=am762d
```

The input data for the restart is TPL file am762r.dat. The “dbs” keyword is set to “am762d”. The following is sample input for the am762r.dat file:

```
RESTART VERSION = 1 $ RESTART FROM AM762D
$ DBS=AM762D SPECIFIED WHEN JOB SUBMITTED
ID UGS, AM762R $ JFC 30S3088
TIME 2
SOL 101
CEND
TITLE = EXAMPLE: RESTART, ATTACH DATABASE VIA DBS=AM762D     AM762R
SUBTITLE = RESTART WITH LARGER LOAD
SELG = ALL $ GENERATE NEW LOAD
SELR = ALL $ REDUCE NEW LOAD
LOAD = 11
DISPLACEMENT = ALL
ELFORCE = ALL
BEGIN BULK
FORCE,11,20,,100.,1.,.8,1.
ENDATA $ AM762R
```

The existing DBALL and MASTER DBsets created in your directory by the “am762d” job are used. The output files from this job are “am762r.f04”, “am762r.f06”, and “am762r.log”.

Using the ASSIGN Statement

This section contains two examples using the ASSIGN statement. The first example, TPL file am763d.dat shows how to use the ASSIGN statement to create the database files. The second example shows how to use the ASSIGN statement to assign database files in a restart job.

```
ASSIGN ’MASTER=DBSDIR:am763d.MYMASTER’
ASSIGN ’DBALL=DBSDIR:am763d.MYDBALL’
$
$ DBSETS CREATED WITH DIRECTORIES AND NAMES AS ASSIGNED ABOVE.
$ THIS IS ALTERNATE METHOD TO BE USED INSTEAD OF SPECIFYING DBS = AM763D
$ WHEN JOB IS SUBMITTED.
$
ID UGS, AM763D $ FILENAME CHANGED 16SEP88 -- JFC
TIME 2
SOL 101 $ STRUCTURED SUPERELEMENT STATICS WITH AUTO RESTART
CEND
TITLE = EXAMPLE: DATABASE CREATED VIA ASSIGN CARDS          AM763D
SUBTITLE = COLD START.
LOAD = 11
DISPLACEMENT = ALL
ELFORCE = ALL
BEGIN BULK
CBEAM,1,1,10,20,0.,1.,0.
FORCE,11,20,,100.,1.,.8,1.
GRID,10,,0.,0.,0.,,123456
GRID,20,,10.,0.,0.
MAT1,100,1.,.08,.064,,.1
ENDDATA
```

Before you submit this job, create a “dbs” directory in your current working directory and set the DBSDIR environment variable to “dbs” as follows:

```
export DBSDIR=dbs
```

in the Korn shell,

```
setenv DBSDIR dbs
```

in the C-shell, or

```
set DBSDIR=dbs
```

on Windows.

Once the DBSDIR environment variable is set, the job is submitted with the command:

```
nxnr nastran TPLDIR:am763d
```

The DBsets “mydball” and “mymaster” are created in the “dbs” directory with the names “am763d.MYMASTER” and “am763d.MYDBALL” respectively. The output files “am763d.f04”, “am763d.f06”, and “am763d.log” are created in the current working directory.

The second example (TPL file am763r.dat) illustrates a restart that uses the ASSIGN statement:

```
RESTART $ RESTART FROM AM763D, SAVE VERSION 1 ON DATABASE
$ ATTACH AM763D DATABASE WITH ASSIGN COMMANDS BELOW
ASSIGN MASTER='DBSDIR:am763d.MYMASTER'
ID UGS,AM763R $ FILENAME CHANGED 16SEP88 -- JFC
TIME 2
SOL 101
CEND
TITLE = EXAMPLE: RESTART, DATABASE ATTACHED VIA ASSIGN CARDS AM763R
SUBTITLE = RESTART -- ADD STRESS RECOVERY COEFFICIENTS TO PBEAM
LOAD = 11
DISPLACEMENT = ALL
ELFORCE = ALL
STRESS = ALL
BEGIN BULK
$ WITH STRUCTURED SOLUTION SEQUENCES (SOL 101+), ALL BULK DATA IS STORED
$ ON DATABASE.
$ ON RESTART, ONLY INCLUDE ADDITIONAL CARDS OR CHANGED CARDS.
/,6 $ DELETE OLD PBEAM CARD ON DATABASE, ADD STRESS RECOVERY COEFFICIENTS
$ AND REPLACE AS FOLLOWS.
PBEAM,1,100,1.,.08,.064,,.1,,+PBEAM1
+PBEAM1,0.0,0.5,0.0,-0.5,0.3,0.0,-0.3,0.0,+PBEAM2
+PBEAM2,YES,0.5,1.0,.08,.064,,.1,,+PBEAM3
+PBEAM3,0.0,0.5,0.0,-0.5,0.3,0.0,-0.3,0.0
ENDDATA $ AM763R
```

To submit the above file, issue the command:

```
nxnr nastran TPLDIR:am763r
```

The DBsets “am763d.MYMASTER” and “am763d.MYDBALL” created by the previous job in the “dbs” directory are used. The output files “am763r.f04”, “am763r.f06”, and “am763r.log” are created in the current working directory.

Using the INIT Statement

DBSets are created using the INIT statement, which is documented in the File Management Section (FMS) of the *NX Nastran Quick Reference Guide*. For example,

```
INIT DBALL LOGICAL=(DBALL1(2000),DBALL2(300KB))
```

creates and allocates two members DBALL1 and DBALL2 to the DBALL DBSet with a size of 2000 GINO blocks for DBALL1 and a size of 300 kilobytes for DBALL2. The size can be specified either as the number of GINO blocks or as a number followed by one of the following modifiers:

M or Mw	Multiply the size by 1024^2 , round up to a BUFFSIZE multiple.
Mb	Multiply the size by $1024^2/(bpw)$, round up to a BUFFSIZE multiple.
K or Kw	Multiply the size by 1024, round up to a BUFFSIZE multiple.
Kb	Multiply the size by $1024/(bpw)$, round up to a BUFFSIZE multiple.
w	Round the size up to a BUFFSIZE multiple.
b	Divide the size by bpw, round up to a BUFFSIZE multiple.

where bpw is 4 using the LP-64 executable, and 8 using ILP-64 executables. The modifier may be specified using any case combination.

Note

This syntax is similar to, but not the same as, the syntax described in "Specifying Memory Sizes".

5.9 Using the INCLUDE Statement

The INCLUDE statement is used to insert a specified file into the input file. This statement is especially useful when you want to partition your input into separate files. The format is

```
INCLUDE filename
```

or

```
INCLUDE logical-symbol:filename
```

The file name must be quoted in single quotes if the name contains lowercase letters, spaces, commas, or dollar signs on Linux; or spaces, commas, or dollar signs on Windows, for example,

```
INCLUDE 'file name'
```

Specifying the INCLUDE Filename

The filename can be continued, if necessary, on multiple lines of the input file. The filename is obtained from an INCLUDE, RFLATER, or RFINCLUDE statement as follows:

1. The *filename* is built up by concatenating tokens. A token is either a blank- or comma-delimited unquoted word or a quoted string (which can be continued across lines).

2. Token are separated by blanks or commas. The blanks or commas separating the tokens are ignored.

3. Statements may be continued by following the last token on a line by a comma, or specifying an incomplete quoted string (i.e., the closing quote is missing from the line). All trailing blanks on the incomplete quoted string's initial line, all leading and trailing blanks on the incomplete quoted string's intermediate lines, and all leading blanks on the incomplete quoted string's final line are ignored.

4. Comments may be specified after the last *filename* token of a line that is not within an incomplete quoted string. The comment is started with an unquoted dollar sign “$”, and continues to the end of the current line.

5. Only the first 72 columns of a line are scanned, i.e., any characters from column 73 and onward are ignored.

These rules are best explained via some examples.

Note

The following examples contain a mixture of Linux and Windows pathnames. The concepts demonstrated by each example are valid on both systems.

```
include datafile.dat
```

The filename is “DATAFILE.DAT”.

```
include ‘c:\abc\def\ghi.include’
```

The filename is “c:\abc\def\ghi.include”.

```
include ‘/mydir’ /level1 /level2/ ‘myfile.x’
```

The filename is “/mydir/LEVEL1/LEVLEL2/myfile.x”.

```
RFAlter ‘/mydir
 /level1
 /level2
   /level3/mydata’
```

The filename is “/mydir/level1/level2/level3/mydata”.

```
include ‘/proj’    $ Proj Name
 ‘/dept123’   $ Dept Name
 ‘/sect 456’   $ Sect Name
 ‘/joe/flange.bdf’   $ User and File Name
```

The filename is “/proj/dept123/sect 456/joe/flange.bdf”.

```
rfinclude c:\project
 $ A comment line
 ‘\Data Files’ \subdir\this file
```

The filename is “C:/pROJECT\Data Files\SUBDIR\THISFILE”.

The following example illustrate what happens when comments or quotes are incorrectly placed.

```
include ‘TPLDIR:alter.file $ comment
 stmt 2 $ word ‘ $ comment 3 ‘ info
```

The filename is “TPLDIR:alter.file $commentstmt 2 $ word ”.

```
include ‘/proj, $ Proj Name
 ‘/dept123, $ Dept Name
 ‘/sect456, $ Sect Name
 ‘/myfile.dat $ File Name
```

The filename is “/proj, $ Proj Name/DEPT123/sect 456, $ Sect Name/MYFILE.DAT”.

Locating INCLUDE Files

Once the *filename* has been obtained from the include statement and any logical symbols have been expanded, up to four filenames on Linux systems and two filename on Windows systems will be searched for. The filename are:

1. The *filename* as specified by the include statement. If *filename* does not end in the the file type specified by the “jidtype” keyword, it is appended.

2. Linux: The *filename* constructed immediately above, converted to lower-case, unless *filename* is already all lower-case (i.e., it was specified as a quoted string).

3. The *filename* as specified by the include statement, without the file type specified by “jidtype”.

4. Linux: The *filename* specified above, converted to lower-case, unless *filename* is already all lower-case (i.e., it was specified as a quoted string).

For example, consider the statement

```
include File1
```

and assume “jidtype=dat” was specified or defaulted. NX Nastran will consider the following filenames on Linux in the order specified:

```
FILE1.dat
file1.dat
FILE1
file1
```

and the following filenames on Windows in the order specified:

```
file1.dat
file1
```

Note

Character-case is insignificant with Windows file names.

For another example, consider the statement

```
include ‘File1.bdf’
```

and assume “jidtype=dat” was specified or defaulted. NX Nastran will consider the following filenames on Linux in the order specified:

```
 File1.bdf.dat
file1.bdf.dat
File1.bdf
file1.bdf
```

and the following filenames on Windows in the order specified:

```
File1.bdf.dat
File1.bdf
```

If *filename* contains a directory component, the software attempts to locate one of the four Linux or two Windows filenames in the specified directory. If none of the names exist or are not readable, a UFM will be issued and the job will exit.

If *filename* does not contain a directory component, the default directory is the current working directory (i.e., the directory where the nastran command was run). If none of the file names exist in the current working directory, NX Nastran looks in the directory containing the file that specified the INCLUDE statement. If none of the file names exist in that directory, and the file that contained the INCLUDE statement was itself included, i.e., the INCLUDE was nested, the directory containing the parent file will be searched.

This nesting will continue until the directory containing the input data file has been searched. If a file has not yet been located, the list of directories specified by the “jidpath” keyword will be searched in order. If no file can be found in any of these directories, a UFM will be issued and the NX Nastran job will exit.

5.10 Using the SSS Alter Library

The SSS Alter directory, *install_dir*/nxn*r*/nast/misc/sssalter on Linux and *install_dir*\nxn*r*\nast\misc\sssalter on Windows contains alters (modifications to NX Nastran solution sequences) and associated support files that represent client-requested or prototype features that are not yet implemented in the standard solution sequences. These alters can be inserted using the INCLUDE statement and the SSSALTERDIR symbol. For example,

```
INCLUDE 'SSSALTERDIR:zfreqa.dat'
```

Included in the SSS Alter directory is the file “README.txt” containing a description of the contents.

5.11 Resolving Abnormal Terminations

NX Nastran generates a substantial amount of information concerning the problem being executed. The .f04 file provides information on the sequence of modules being executed and the time required by each of the modules; the .log file contains system messages. A list of known outstanding errors for the current version is delivered in the file *install_dir*/nxn*r*/nast/doc/error.lis on Linux and *install_dir*\nxn*r*\nast\doc\error.lis on Windows. Please consult this file for limitations and restrictions.

NX Nastran may terminate as a result of errors detected by the operating system or by the program. If the DIAG 44 is set (see the diag keyword and the *NX Nastran Quick Reference Guide*), NX Nastran will produce a dump of several key internal tables when most of these errors occur. Before the dump occurs, there may be a fatal message written to the .f06 file. The general format of this message is

```
***SYSTEM FATAL ERROR 4276, subroutine-name ERROR CODE n
```

This message is issued whenever an interrupt occurs that NX Nastran is unable to satisfactorily process. The specific reasons for the interrupt are usually printed in the .f06 and/or .log file; “n” is an error code.

Whenever the System Fatal Error 4275 or 4276 is associated with a database error, further specific information is written to the .f06 file as follows:

```
bio-function ERROR - STATUS = errno, FILX = i, LOGNAME = logical, NSBUF3 = j
FILE = filename
BLKNBR = k
ERROR MESSAGE IS --
```

```
error-message-text
```

The FILE and/or BLKNBR lines may not be present, depending upon the bio-function issuing the message.

Terminating a Job

There may be instances when a running job must be prematurely terminated; this is accomplished using one of the following procedures:

Job Running in the Foreground (batch=no on Linux ; all jobs on Windows)

Use the interrupt sequence (on Silicon Graphics systems this sequence is usually “Ctrl-\”; on other systems “Ctrl-C”).

Job Running in the Background (batch=yes or after=time on Linux)

Use the “ps” command to find the process ID (PID) of the NX Nastran job (i.e., the *install_dir*/nxn*r*/*arch*/analysis executable) and issue the command

```
kill pid
```

where *pid* is the process ID.

Job Running Under NQS or NQE (queue=queue_name on Linux)

1. Use “qstat -a” to find the request-id of your job.

2. Use “qdel *request-id*” to delete a job that has not yet started; or use “qdel -k *request-id*” to kill a job that has already started where request-id is the request ID.

Flushing .f04 and .f06 Output to Disk

As NX Nastran writes to the .f04 and .f06 files, the FORTRAN runtime libraries will buffer this I/O in memory to reduce the amount of time consumed by disk I/O. When the buffers are filled (i.e., NX Nastran has written a sufficient amount of information to the .f04 or .f06 file), the buffers will be flushed to the files by the FORTRAN runtime libraries. In a large job, some modules may do substantially more computation than I/O. As a result, the I/O may remain in the FORTRAN buffers (possibly for several hours) before they are written to disk.

To do asynchronous flushing of the .f04 and .f06 files, enter the command

```
kill -USR1 pid
```

where *pid* is the process ID of the running NX Nastran job (i.e., the *install_dir*/nxn*r*/*arch*/analysis executable). There may be a time delay between the time you issue the kill command and time the files are actually updated.

Common System Errors

The most common system errors encountered during an NX Nastran job are described below.

Disk I/O Errors

- ERRNO 1 (EPERM) - no permission to file (all systems).

  Check the ownership and mode of the file or directory with the “ls -l” command. Change either the ownership or permissions of the file or the directories along the path. The chgrp(1) command is used to change the group of a file, chmod(1) is used to change permissions of the file, and chown(1) is used to change ownership of the file.

- ERRNO 27 (EFBIG) - file is too large (all systems)

  This error occurs if a file's size exceeds a resource limit. The resource limits in effect during the job's execution are printed in the .log file under the heading “Current Resource Limits.” Increase the “-lf” and “-lF” parameters on your qsub command if you are running NQS or NQE; ask your system administrator to increase your “File Size” limit.

- ERRNO 28 (ENOSPC) - disk space is completely filled (all systems).

  NX Nastran deletes its scratch files at termination even if the disk space fills up. Therefore, the df(1) command may show a large amount of free space even though the job failed due to lack of disk space. Both the current working directory and the scratch directory need to be checked. Move your files to a disk with more space (see the “out”, “dbs”, and “sdirectory” keywords), or delete unnecessary files from the disk.

Inability to Allocate the Requested Amount of Memory (OPEN CORE Allocation Failed)

- Temporary lack of swap space.

  This error may be caused by too many processes running at the same time. Decrease the number of processes or increase the available swap space.

- The data segment of the process has exceeded the resource limit.

  The resource limits in effect during the job's execution are printed in the .log file under the heading “Current Resource Limits.” Ask your system administrator to increase your “Data Segment Size” (all) or “Virtual Address Space”.

It may also be possible to correct these errors with the following:

- Reduce the amount of memory requested by the “memory” keyword.

- Increase the “-lm” and “-lM” parameters if you directly submitted your job to NQS or NQE using a “qsub” command.

- Increase the “prmdelta” or “ppmdelta” keyword values if you submitted your job to NQS or NQE using the nastran command’s “queue” keyword

Chapter 6: Using the Advanced Functions of NX Nastran

This chapter discusses the NASTRAN statement, as well as how to manage NX Nastran's internal memory allocations and databases. It also shows how to interpret how to run a job on a remote system, run a DMP job, use the ISHELL module, and finally, how to create alternate delivery databases.

6.1 Using the Advanced Keywords

The following is a partial list of the advanced keywords that may be used on the command line or placed into RC files as appropriate. More basic keywords are listed in "Using the Basic Keywords"; keywords specific to remote processing are listed in "Running a Job on a Remote System (Linux)" . Finally, a complete list of all keywords and their syntax is listed in "Keywords".

All Systems

Keyword	Purpose
buffsize	Specifies the size of database I/O transfers.
bpool	Specifies the number of GINO blocks set aside for buffer pooling.
delivery	Specifies an alternate delivery database name.
exe	Specifies an alternate solver executable.
nastran	Specifies NASTRAN statements.
proc	Specifies an alternate solver executable file type.
rank	Specifies the rank size for the sparse solvers.
smem	Specifies the memory to set aside for MEMFILE portion of the SCRATCH DBSet.
sysfield	Specifies global SYS parameters. See "Using the SYS Field".
sysn	Specifies SYSTEM cell values.
post	Specifies commands to be executed after the job completes.
pre	Specifies commands to be executed before the job begins.

Queuing (Linux)

Note

These capabilities are dependent upon the queue submission commands defined by the "submit" keyword and your queuing system. The keywords may not work on your system.

Keyword	Purpose
ppcdelta	Specifies the per-process CPU time limit delta.
ppmdelta	Specifies the per-process memory limit delta.
prmdelta	Specifies the per-request memory limit delta.
qclass	Specifies an optional queue class.
qoption	Specifies other queue command options.
submit	Defines queues and their associated submittal commands.

6.2 Using the NASTRAN Statement

The NASTRAN statement allows you to change parameter values at runtime.

The format of NASTRAN statements is

```
NASTRAN KEYWORD1=A, KEYWORD2=B, ... KEYWORDi=I
```

An input file may contain more than one NASTRAN statement. A full description of these keywords is found in “The NASTRAN Statement” in the *NX Nastran Quick Reference Guide*”. A brief description of a few of the keywords follows:

AUTOASGN

AUTOASGN is used to determine which DBsets are automatically assigned (see the following table). The default is AUTOASGN=7, which specifies that all DBsets are to be automatically assigned.

Value	Default DBsets	Delivery DBsets	DBLOCATEd DBsets
0			
1	X		
2		X	
3	X	X	
4			X
5	X		X
6		X	X
7 (Default)	X	X	X

Note

- Default DBsets are the user-default DBsets and any DBsets specified by INIT statements (see Table 5-6).
- Delivery DBsets contain the Structured Solution Sequences.
- DBLOCATEd DBsets are the DBsets specified by DBLOCATE statements. See “DBLOCATE” in the *NX Nastran Quick Reference Guide*.

BUFFPOOL, SYSTEM(114)

See the “bpool” command line keyword, (bpool).

BUFFSIZE, SYSTEM(1)

See the “buffsize” command line keyword, (buffsize).

PARALLEL, SYSTEM(107)

See the parallel command line keyword.

SYSTEM(128)

SYSTEM(128) specifies the maximum interval of CPU time (in minutes) between database directory updates to the MASTER DBSET when the INIT MASTER(RAM) option is being used. The default is 5 minutes. See “DBUPDATE” in the *NX Nastran Quick Reference Guide* for more information.

SYSTEM(198), SYSTEM(205)

See the rank keyword.

SYSTEM(207)

See the lock SYS field keyword.

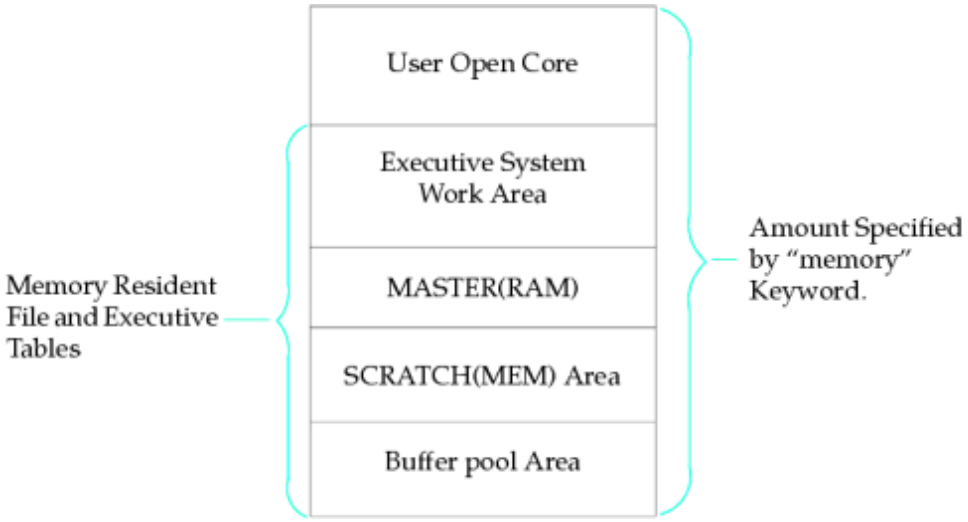
SYSTEM(275)

SYSTEM(275) sets the time-out for an ISHELL program to complete its work. If the value is negative (the default is -1), the ISHELL module will wait until the executable finishes, i.e., there is no time-out. If the value is positive, the ISHELL module will wait for the specified number of seconds.

If the value is zero, the ISHELL module will determine if an executable can be found, and return a zero status if found and a non-zero status if it can’t be found.

6.3 Managing Memory

Memory is dynamically allocated at runtime with the “memory” keyword of the nastran command. The memory can be partitioned in a variety of ways (see the memory map at the top of the .f04 file for the actual memory allocation used in a job). To make the most effective choice of the sizing parameters, see the following map of NX Nastran’s memory:



As can be seen in this diagram, the memory available for use by NX Nastran modules (user open core) is the amount specified by the “memory” keyword (open core size) less the space required by memory resident files and executive tables. The actual user open core is calculated as follows:

UserOpenCore = MEM – (EXEC + RAM + SMEM (when defined as GINO blocks) X BUFFSIZE + BUFFPOOLX (BUFFSIZE + 10))

MEM	The total size of open core. There is no default. Set by the memory keyword.
EXEC	The executive system work area. The size is 70409 + 4 x *BUFFSIZE* words using the LP-64 executable, and 70409 + 8 x *BUFFSIZE* words using ILP-64 executables.
RAM	NDDL tables. The default is 30000. Set by the FMS statement INIT MASTER (RAM=*value*).
SMEM	The memory-resident file space for temporary database files. The default is 100. Set by the FMS statement INIT SCRATCH (MEM=value) or the smemory keyword.
BUFFSIZE	The maximum BUFFSIZE used for all the DBsets referenced by the job. The default is 8193. Set by the buffsize keyword.
BUFFPOOL	The buffer pool area for permanent database files. The default size 37. Set by the bpool keyword.

The INIT statement may be used to size MASTER and SCRATCH memory. Several examples of the INIT statement, along with an explanation of their uses, follow:

1. If the available memory is a critical resource, then using the following selection reduces memory requirements at the expense of increased CPU and wall-clock time.

```
INIT SCRATCH(NOMEM)                $ temporary database files
```

2. Performance gains may be made by increasing the memory-resident area for the scratch and permanent DBset(s) as follows. Note that the default RAM is sufficiently large and need not be increased.

```
NASTRAN BUFFPOOL=70    $ increase permanent DBSets
INIT SCRATCH (MEM=200)     $ increase scratch memory
```

3. If disk space is critical, then all DBsets may be deleted at the end of the job by specifying “S” on the INIT MASTER statement as follows:

```
INIT MASTER(S) $ delete DBsets at end of job
```

   This statement is identical to specifying “scratch=yes” on the command line.

4. If disk space is critical, but data recovery restarts are required, then a database may be created that will support data recovery restarts by setting “scratch=mini” on the command line.

```
nxnr nastran example scratch=mini
```

6.4 Managing DBSets

Using the SYS Field

The SYS field is used to specify computer-dependent parameters on ASSIGN statements. If your computer does not recognize a particular parameter, it is silently ignored. This keyword is specified as a comma separated list of keyword=value pairs. For example, file locking may be disabled on for a particular DBset with the following statement:

```
ASSIGN 'DBALL=mydball.DBALL' SYS=LOCK=NO
```

A global SYS field for all DBsets can be specified by the sysfield keyword.

The following tables describe the SYS field parameters. A complete description of parameters and their syntax is available in "SYS Parameter Keywords".

All Systems

Keyword	Purpose
lock	Lock database files.

Systems Supporting File Mapping

(See Table 5-7)

Keyword	Purpose
mapio	Use the virtual memory system to map database files to memory.
wnum	Specifies the default number of maps used on database files.
wsize	Specifies the default size of maps used on database files.

Systems Supporting Buffered I/O

(See Table 5-7)

Keyword	Purpose
buffio	Uses intermediate buffers to hold database file records
wnum	Specifies the default number of buffers used for database files
wsize	Specifies the default size of buffers used for database files

Using File Mapping

File mapping is a way to tell the operating system to use the virtual paging system to process a file. From the perspective of the process, file mapping effectively changes the file I/O operations from synchronous to asynchronous because the paging functions of the operating system perform the I/O as part its normal virtual memory management. File mapping can be used for both permanent and temporary DBsets. See Table 5-7 to determine if file mapping is available on your computer.

The "wsize" and "wnum" parameters, described in "SYS Parameter Keywords", specify the size of the window mapping the file to memory and the number of windows or maps that will be used for each file. The larger the window, the less often it must be moved when the file is sequentially read or written. Multiple maps allow several I/O streams to be active in the same file.

File mapping is controlled using the ASSIGN statement SYS field for individual DBsets and, globally, using the "sysfield" command line keyword, (sysfield).

As an example, if file mapping is to be enabled for all files, the "sysfield" keyword in the command initialization or RC file or on the command line is:

```
sysfield=mapio=yes
```

If file mapping is to be disabled for all files, the "sysfield" keyword is:

```
sysfield=mapio=no
```

If file mapping is to be enabled for all but a specified set of DBsets, both “sysfield” keyword and ASSIGN specifications are required. In the command initialization file, RC file, or on the command line, specify:

```
sysfield=mapio=yes
```

and, in the NX Nastran data file, specify:

```
ASSIGN logical-name=filename,SYS=MAPIO=NO
```

for those files to be processed using normal disk I/O processing.

If file mapping is to be disabled for all but a specified set of DBsets, both “sysfield” keyword and ASSIGN specifications are required. In the command initialization file, RC file, or on the command line, specify:

```
sysfield=mapio=no
```

and, in the NX Nastran data file, specify:

```
ASSIGN logical-name=filename,SYS=MAPIO=YES
```

for those files to be processed using file mapping.

Using Buffered I/O

Note

1. See Table 5-7 to determine if buffered I/O is available on your computer.

Buffered I/O instructs NX Nastran to “buffer” or use intermediate memory areas to hold records of a file before either writing them out to disk or copying them to the NX Nastran internal areas. The primary purpose for using buffered I/O is to increase data reuse and, in some cases, to increase the actual read/write data lengths beyond that normally used by NX Nastran. Buffered I/O can be used for both permanent and temporary DBSETS.

The wsize and wnum parameters specify the size of the buffer to be used to hold file records and the number of such buffers to be used. The larger the buffer, the less often actual physical read/write operations are needed when the file is sequentially read or written. Multiple buffers allow several I/O streams to be active in the same file.

Buffered I/O is controlled using the ASSIGN statement SYS field for individual DBsets and, globally, using the “sysfield” command line keyword. These are described in “Using the SYS Field”.

As an example, if buffered I/O is to be enabled for all files, the “sysfield” keyword in the command initialization or RC file or on the command line is:

```
sysfield=buffio=yes
```

If buffered I/O is to be disabled for all files, the “sysfield” keyword is:

```
sysfield=buffio=no
```

If buffered I/O is to be enabled for all but a specified set of DBsets, both “sysfield” keyword and ASSIGN specifications are required. In the command initialization file, RC file, or on the command line, specify:

```
sysfield=buffio=yes
```

and in the NX Nastran data file, specify:

```
ASSIGN logical-name=filename,SYS=BUFFIO=NO
```

for those files to be processed using normal disk I/O processing.

If buffered I/O is to be disabled for all but a specified set of DBsets, both “sysfield” keyword and ASSIGN specifications are required. In the command initialization file, RC file, or on the command line, specify:

```
sysfield=buffio=no
```

and in the NX Nastran data file, specify:

```
ASSIGN logical-name=filename,SYS=BUFFIO=YES
```

for those files to be processed using buffered I/O.

Interpreting Database File-Locking Messages (Linux)

All database files are locked using the operating system function “fcntl(2)”. This prevents two or more NX Nastran jobs from interfering with one another; however, this does not prevent any other program or operating system command from modifying the files.

A read-write (exclusive) lock is requested for every database file that is to be modified. A read-only (shared lock) is requested on every database file that is not modified, e.g., DBLOCATEd databases. If the lock request is denied because another NX Nastran job is using the file in a potentially conflicting manner, the following fatal error message is written to the .f06 file:

```
bio-function ERROR - STATUS = errno, FILX = i, LOGNAME = logical, NSBUF3 = j

FILE = filename

ERROR MESSAGE IS --

Unable to acquire a lock_type lock.

lock-type-explanatory-text

Process ID pid is holding a conflicting lock.
```

where *lock-type-explanatory-text* is:

- lock_type is “read-only”:

```
This operation failed because another process already
holds a read-write lock on this file.
```

- lock_type is “read-write”:

```
This operation failed because another process already
holds a read-write or read-only lock on this file.
```

Some systems will deny a file lock because of an internal resource limit. In these cases, the job is allowed to continue, and the following message will be written to the .f06 file:

```
bio-function ERROR - STATUS = errno, FILX = i, LOGNAME = logical, NSBUF3 = j

FILE = filename

ERROR MESSAGE IS --

Unable to acquire a lock_type lock.

computer—specific-text
```

```
advisory-text
```

where *computer-specific-text* is:

All systems
```
The system wide maximum number of file locks has been exceeded.
 See ENOLCK in man 2 fcntl.
```

and *advisory-text* is:

- *lock_type* is “read-only”

```
If another job modifies this file during this run, there is the potential
for incorrect results to occur in this job.
```

- *lock_type* is “read-write”

```
If another job accesses this file during this run, there is the potential
for the file to be damaged and/or incorrect results to occur in both jobs.
```

Disabling File Locking

File locking can be disabled by:

- Setting “sysfield=lock=no” in an RC file or on the command line; see “Using the Advanced Keywords”. This affects all DBsets in the job.
- Setting SYSTEM(207) to a nonzero value using the NASTRAN statement; see “Using the NASTRAN Statement”. This affects all DBsets in the job.
- The following informational message is written to the .f06 file:

```
*** SYSTEM INFORMATION MESSAGE - BIO
SYSTEM(207).NE.0 - File locking suppressed.
```

- Setting SYS=LOCK=NO on an FMS INIT statement; see “Using the SYS Field” . This only affects the specific DBset (s).

6.5 Running a Job on a Remote System (Linux)

The nastran command offers a mechanism to run simple jobs on a computer other than the computer you are currently logged onto via the node keyword. In the descriptions that follow, the “local” node is the computer you issue the nastran command on, the “remote” node is the computer named by the “node” keyword, i.e., where the NX Nastran analysis will run.

Following are some general requirements for running remote jobs:

1. NX Nastran must be properly installed on the remote system.
2. The input data file must be accessible on the local host.
3. INCLUDE files must be local-to, or visible-from, the remote system.
4. All default output files, i.e., those without ASSIGN statements, will be written to a directory accessible to the local host.

5. You must have “remote execution” privileges on the remote system. That is, a password must not be required to execute a remote copy (rcp) or remote shell (rsh or remsh) command. See your system administrator for information on this.

6. In a restart, i.e., a job that uses an existing database, the DBSets must be local-to, or visible-from, the remote system.

7. You must have r-command access to each system you want to access in a remote job.

   You can test this with the following command:

   ```
   rsh <node> [-l <username>] date
   ```

   where “<node>” is the name of the node and “<username>” is an alternate username on the remote system if your current username is not valid. For example:

   ```
   rsh node 1 date
   ```

   The output from the above command should be a single line containing the current date on node 1 in a format similar to:

   ```
   Thu Sep 30 13:06:49 EDT 2005
   ```

   If any other output is present, determine the source of the output and correct the problem. If you cannot eliminate the output, you will not be able to use the remote execution capabilities of the nastran command.

   Note

   Recall that remote executions do not run a “login” shell. That is, your “.profile” or “.login” script is not executed.

   This capability does not permit Windows systems to run jobs on other Windows computers.

   There are some circumstances where Windows “rsh” and “rcp” commands do not perform reliably. We have found that problems are more likely to occur on high-speed networks.

When running a remote job, nastran keywords are processed on both the local and remote systems. Keywords that control the job’s output and interaction with you are processed on the local system. These are:

Table 6-1. Remote Processing Keywords

Keyword	Purpose
append	Requests the .f06, .f04, and .log files to be concatenated.
batch	Requests the job is to be run in the background.
delete	Unconditionally deletes files after job completes.
ncmd	Specifies an alternate notification command.
node	Specifies the node the job will be processed on.
notify	Requests notification when the job completes.
old	Specifies versioning or deletion of previously existing output files.
oldtypes	Specifies additional user file types to be versioned or deleted.
out	Specifies an alternate output file prefix.
rcmd	Specifies the nastran command path on the remote system.

scratch	Specifies the database DBSets are to be deleted at job completion.
sdirectory	Specifies the directory to contain NX Nastran temporary files.
trans	Requests translation of the .xdb file.
username	Specifies an alternate username on the remote host.
xmonitor	Requests XMONITOR to monitor the job's progress.

The "sdirectory" keyword is special, as the command line, RC files on the current host, and RC files on the remote host will all be considered when establishing a scratch directory. All remaining keywords are only scanned on the remote Linux system and must specify Linux pathnames.

Once "node=*remotenode*" is processed, the following processing takes place:

1. Process the RC files on the local system if the "version" keyword has been defined in the command initialization file or the command line.

2. Process the RC file specified by the "rcf" keyword if it was defined on the command line.

3. Determine the full pathname of the input file so that its visibility from *remotenode* can be tested.

4. Create a "touch" file in the specified output file so that its visibility from *remotenode* can be tested.

5. If the "rmtdeny" utility, i.e., *install_dir*/nxn*r*/*arch*/rmtdeny, exists and is executable, run it and examine its output. If *remotenode* is listed, display an error and cancel the job.

6. If the "rmtaccept" utility, i.e., *install_dir*/nxn*r*/*arch*/rmtaccept, exists and is executable, run it and examine its output. If *remotenode* is not listed, display an error and cancel the job.

7. Ensure "scratch=no" was set if the "dbs" keyword was set.

8. Verify that *remotenode* exists and you are able to run a command on that system.

9. If the "rcmd" keyword was specified, attempt to execute that command on *remotenode*, display an error and cancel the job if it fails. Otherwise, attempt to execute the pathname of the current nastran command on *remotenode*. If it fails, attempt to run the basename of the current nastran command on *remotenode*. Display an error and cancel the job if both checks fail.

10. Run the remote nastran command identified in the previous step to determine: if the input data file is visible; if the "touch" file is visible, if the "sdirectory" (if identified on the local system) exists; if the "dbs" directory (if identified on the local system) exists; the "sdirectory" value in the RC files defined on the remote system; and finally the numeric format of the remote system.

11. Display an error and cancel the job if a scratch directory was identified on the command line or in a local RC file, but does not exist on the remote node.

12. Display an error and cancel the job if the "dbs" directory was identified on the command line or in a local RC file, but does not exist on the remote node.

13. Delete the "touch" file created above.

14. Make sure a RECEIVE executable exists on the local node if "trans=yes" was specified, or "trans=auto" was specified and the numeric formats of the local and remote nodes differ.

15. The remaining steps are done in a background process (possibly some time later) if “batch=yes” or “after” was specified.

    a. Copy the input data file to the scratch directory if the remote host could not see the input data file.

    b. Set “out” to the scratch directory if the remote host could not see the output directory.

    c. Copy the remaining keywords on the command line that were not processed, to a local RC file in the scratch directory on the remote node.

    d. Run the job on the remote node.

    e. Process the “old” and “oldtypes” keywords on the local node.

    f. Copy the output files (.f04, .f06, .log, .ndb, .pch, .plt) to the directory specified by the “output” keyword and delete the files from the remote node if the output directory was not visible from the remote node.

    g. Process the “append” keyword on the local node.

    h. Run the RECEIVE program if required by the “trans” keyword.

    i. Process the “notify” keyword on the local node.

Once the job has completed, the .f06, .f04, .log, .ndb, .op2, .plt, .pch, and .xdb files will be present as if the job were run locally. Binary files, i.e., .op2 and .plt, will only be usable on the local node if the local and remote nodes use the same numeric format. The .xdb file will be translated via TRANSMIT and RECEIVE unless “trans=no” was specified.

Note

No attempt is made to copy DBSet files between the local and remote systems. If this is required, you must handle this yourself and set the “dbs” keyword as required.

Several examples are provided.

```
nxnr nastran example node=othernode batch=no
```

This job will run on node “othernode”. The .f04, .f06, .log, .pch, .plt, and .xdb files will be brought back to the current node as if the job were run locally.

```
nxnr nastran example node=othernode rcmd=/some/path/bin/nxnr
```

This job will also run on “othernode”, but the path to the nastran command has been specified explicitly.

```
nxnr nastran example node=othernode dbs=/dbs
```

This job will also run on “othernode”, but will use the “/dbs/example.*” DBSet files. These files must exist on “othernode” prior to running this command if this is a restart job. Once the job completes, the DBSet files will be left as is.

```
nxnr nastran example node=uxsrv sdir=/tmp
```

This example will run a job on Linux node “uxsrv” using the nastran command in the default PATH with all scratch files residing in /tmp. Note that the “sdir” keyword could have been set in an RCF file.

```
nxnr nastran example node=uxsrv sdir=
```

This job will use the default scratch directory on “uxsrv”.

```
nxnr nastran example node=uxsrv rcmd=/siemens/bin/nxnr sdir=
```

This job will use the nastran command /siemens/bin/nxn*r* on “uxsrv”.

6.6 Running an ISHELL Program

The ISHELL module allows you to invoke your own program from DMAP to perform custom processing. Two features are provided to make running your program easier.

The first feature is the ability to construct a full named based on the up-to eight character name provided by DMAP and a list of file-type associations. NX Nastran will first attempt to find an executable in the current directory using the name as-is from the DMAP call, i.e., all upper-case. On Linux, if this name cannot be found, another attempt is made by converting the name to all lower-case.

If a name was not found, the Command Processor Associations defined by the “ishellext” keyword will be used to construct additional names by concatenating the DMAP name with each file-type in turn until the name is found or the table is exhausted. The command processor extensions consist of pairs of file-types and commands. On Linux systems, the default command processor associations are:

File-Type	Command Processor
null	*directly execute*
.sh	sh
.ksh	ksh
.csh	csh
.pl	perl
.prl	perl

On Windows, the default command processor associations are:

File-Type	Command Processor
.bat	*directly execute*
.exe	*directly execute*
.com	*directly execute*
.pl	perl
.prl	perl

Note

While this capability is similar to the Windows “File Type Associations,” it does not use that information.

These tables are processed in the order shown.

If none of the names exist in the current working directory, NX Nastran will resort to the second feature design to assist in using the ISHELL module, the “ishellpath” keyword. If this keyword is set, NX Nastran will repeat the search described above for each of the directories listed by the keyword. To aid in using this keyword, the nastran command will set the default value for “ishellpath” as the directory containing the input data file if you have not set the keyword on the command line, via the NXN_ISHELLPATH environment variable, or in an RC file.

If a file has still not been found in either the current working directory or any of the directories listed by the “ishellpath” keyword, the system PATH will be searched. Finally, if a suitable file was not found, a UFM will be issued.

A sample ISHELL job is provided by the files TPLDIR:qaishell.dat, TPLDIR:QAISHELL, and TPLDIR:qaishell.pl. The ISHELL call is:

```
.
.
.
ISHELL //'QAISHELL'/S,N,IRTN/
NOINT/NOREAL/NOCMPX/NOCHAR/NOUNIT/
INT1/INT2/INT3/INT4/
REAL1/REAL2/REAL3/REAL4/
CMPL1/CMPL2/CMPL;3/CMPL4/
STRING1/STRING2/STRING3/STRING4/
/UNIT1/UNIT2/UNIT3/UNIT4 $
.
.
.
```

For the following example, assume the nastran command provides the default value for the “ishellpath” keyword, i.e., the directory containing the input data file.

```
nxnr nastran qaishell
```

On Linux, the following names will be checked (assuming the default command processor associations): QAISHELL, qaishell, QAISHELL.sh, qaishell.sh, QAISHELL.ksh, qaishell.ksh, QAISHELL.csh, qaishell.csh, QAISHELL.pl, qaishell.pl, QAISHELL.prl, and finally qaishell.prl. Since the file “QAISHELL” exists in the same directory as the input file, it will be found after first looking for the names in the current working directory.

On Windows, the following names will be checked (assuming the default command processor associations): QAISHELL.BAT, QAISHELL.EXE, QAISHELL.COM, QAISHELL.PL, and finally QAISHELL.PRL. Since the file “qaishell.pl” exists in the same directory as the input file, it will be found after first looking for the names in the current working directory.

Defining Command Processor Associations

The nastran command treats each specification of the “ishellext” keyword as either an addition to, modification of, or deletion from, the current definition. For example, using the default command processor associations, specifying

```
ishellext=tcl=wish
```

will add a new processor, “wish”, for the file-type “.tcl”, after the last currently defined processor. Specifying

```
ishellext=pl=
```

will delete the current association of “perl” for the file-type “.pl”. Finally,

```
ishellext=sh=ksh
```

will replace the “sh” definition for the “.sh” file type on Linux.

To change the processing order, delete the current entry and then respecify it (to append it to the end of the table). For example, to force Linux systems to find “qaishell.pl” before “QAISHELL”, specify

```
ishellext=.=,.=''
```

Note that this first deletes the null processor “.=”, and then re-specifies it as “.=””.

```
ishellext=.='',sh=sh,ksh=ksh,csh=csh,pl=perl,prl=prl
ishellext=bat='',exe='',com='',pl=perl,prl=perl
```

These two examples are the default associations for Linux and Windows respectively.

Special Considerations (Windows)

On Windows, all executable files must have a non-null file type; this is why the “QAISHELL” script cannot be used on Windows, even if you have a Korn shell installed.

Finally, you can use a hash mark, “#”, in place of the equals sign on Windows to facilitate setting the processor association in a “.bat” file. For example,

```
ishellext#bat#'',exe#'',com#'',pl#perl,prl#perl
```

is an alternate definition of the default Windows association.

6.7 Improving Network File System (NFS) Performance (Linux)

The Network File System (NFS) is software allowing file systems on remote computers to appear as if they were mounted on the local computer. There are two daemons that handle NFS traffic: “nfsd” handles file system access requests by the local computer to remotely mounted file systems; “biod” handles requests by remote computers to access local file systems.

These daemons have been designed so that multiple executing copies of each daemon increase NFS traffic capacity. Two of the possible causes of poor NFS performance are a lack of sufficient daemons to handle NFS requests made by the local computer to remotely mounted file systems (nfsd), or a lack of sufficient daemons to handle NFS requests of local file systems by remote computers (biod). The default number of daemons for nfsd and biod is typically four of each. This default is usually fine for a stand alone workstation used by one person. If you or others are accessing many remote file systems or run many NX Nastran jobs accessing file systems on file servers or remote workstations, you may need to increase the number of nfsd and biod daemons on both systems to increase NFS performance.

If you are running three or more NX Nastran jobs accessing disks on remote computers via NFS, you should increase both nfsd and biod daemons above the standard defaults. A good starting point is twelve (12) nfsd daemons and eight (8) biod daemons per CPU on client and server computers, respectively.

Your system administrator can change both system’s configurations to start additional NFS daemons. The administrator can also monitor network statistics with “nfsstat” to ensure network traffic is being handled efficiently. Additional daemon tuning may be necessary for your specific network needs.

6.8 Creating and Attaching Alternate Delivery Databases

NX Nastran uses the Structured Solution Sequences (SSS), located in *install_dir*/nxn*r*/*arch* on Linux and *install_dir*\nxn*r**arch* on Windows, to specify the default solution sequences. You may modify and store a tailored solution sequence by creating a new delivery database. This procedure is also useful to eliminate unwanted solutions from the delivery database or add additional solution sequences.

The following files are delivered in the *install_dir*/nxn*r*/nast/del/ directory on Linux and *install_dir*\nxn*r*\nast\del\ on Windows:

Filename	Description
buildsss	Linux script used to build delivery database.
buildsss.bat	Windows BAT file to build delivery database.
*.dat	SubDMAP source.
*.dck	SubDMAP source that must be preprocessed by EDSFPP.
*.ddl	NDDL source.

Using the Supplied Source

To rebuild the delivery database using the supplied source, use the following procedure:

1. Change the working directory to an empty work directory. For example,

   ```
   cd $HOME/new-del
   ```

   on Linux, or

   ```
   cd %HOMEDRIVE%%HOMEPATH%\new-del
   ```

   on Windows.

2. Rebuild the delivery database.

   ```
   nxnr buildsss
   ```

Upon completion of this procedure, the delivery files SSS.MASTERA, SSS.MSCOBJ, and SSS.MSCSOU are created. These files are attached with the “delivery” keyword.

These files may be installed in the master architecture directory (if you have write access) with the command:

```
cp SSS.* install_dir/nxnr/arch
```

on Linux, or

```
copy SSS.* install_dir\nxnr\arch
```

on Windows.

Using Modified Source

To build a modified delivery database, use the following procedure.

1. Change the working directory to an empty work directory. For example,

   ```
   cd $HOME/new-del
   ```

   on Linux, or

```
cd %HOMEDRIVE%%HOMEPATH%\new-del
```

on Windows.

2. Copy the subDMAP and NDDL source files that are to be modified to the current directory.

```
cp install_dir/nxnr/nast/del/subDMAP.dat .
cp install_dir/nxnr/nast/del/subDMAP.dck .
cp install_dir/nxnr/nast/del/nddl.ddl .
```

on Linux, or

```
copy install_dir\nxnr\nast\del\subDMAP.dat .
copy install_dir\nxnr\nast\del\subDMAP.dck .
copy install_dir\nxnr\nast\del\nddl.ddl .
```

on Windows where subDMAP and nddl are the specific files to be modified.

3. Modify the desired subDMAP and/or NDDL source files using a text editor.

4. Rebuild the delivery database.

```
nxnr buildsss src=.
```

Upon completion of this procedure, the delivery files SSS.MASTERA, SSS.MSCOBJ, and SSS.MSCSOU are created. These files are attached with the “delivery” keyword. These files may be installed in the master architecture directory (if you have write access) with the command:

```
cp SSS.* install_dir/nxnr/arch
```

on Linux, or

```
copy SSS.* install_dir\nxnr\arch
```

on Windows.

Chapter 7: Using the Utility Programs

- *Overview*
- *F06 output to Excel*
- *ESTIMATE*
- *F04REPRT*
- *EDSACT*
- *HEATCONV*
- *MSGCMP*
- *NEUTRL*
- *OPTCONV*
- *PLOTPS*
- *RCOUT2*
- *RECEIVE*
- *TRANS*
- *Building the Utilities Delivered in Source Form*

7.1 Overview

This chapter describes how to use the various NX Nastran utility programs. The following table groups these utilities by function

Table 7-1. Utility Program Functions

Utility	Function
F06 output to Excel	On Windows operating systems, the F06 to Excel data translator copies printed output data from an .f06 file into Microsoft Excel.
ESTIMATE	Estimates system requirements of an NX Nastran job and suggests performance improvements.
F04REPRT	Perl script to summarize or compare .f04 files.
HEATCONV	Reformats MSC.Nastran Version 67 heat-transfer and optimization data files into current formats.
OPTCONV	
EDSACT	Accumulates and summarizes NX Nastran accounting data.
MSGCMP	Compiles the message catalog.
NEUTRL	Converts NX Nastran plot files to PostScript or neutral format.
PLOTPS	
RCOUT2	Converts neutral-format OUTPUT2 files to binary format.
RECEIVE	Moves results database (XDB) files between dissimilar computers.
TRANS	

The sections on each utility describe the programs and present applicable keywords and examples. The final section provides instructions on building the source code utilities.

7.2 F06 output to Excel

On Windows operating systems, the NX Nastran F06 to Excel data translator is available to transfer printed output data in an .f06 file into Microsoft Excel. This data translator runs independently of NX Nastran.

The translator is available in the NX Nastran installation hierarchy at

```
installation_directory/nxnr/util/
```

After installing the Perl prerequisites, you can start the translator interface in the following ways.

- In the *util* folder, double click the following command

  You will be prompted to browse and select an .f06 file, and the translator interface will start.

- At the command line prompt, enter the input:

  ```
  installation_directory/nxnr/util/toExcel.pl -g
  ```

  You will be prompted to browse and select an .f06 file, and the translator interface will start.

- At the command line prompt, enter the input:

  ```
  installation_directory/nxnr/util/toExcel.pl -g file.f06
  ```

  Because the input includes the .f06 file, the translator interface will start.

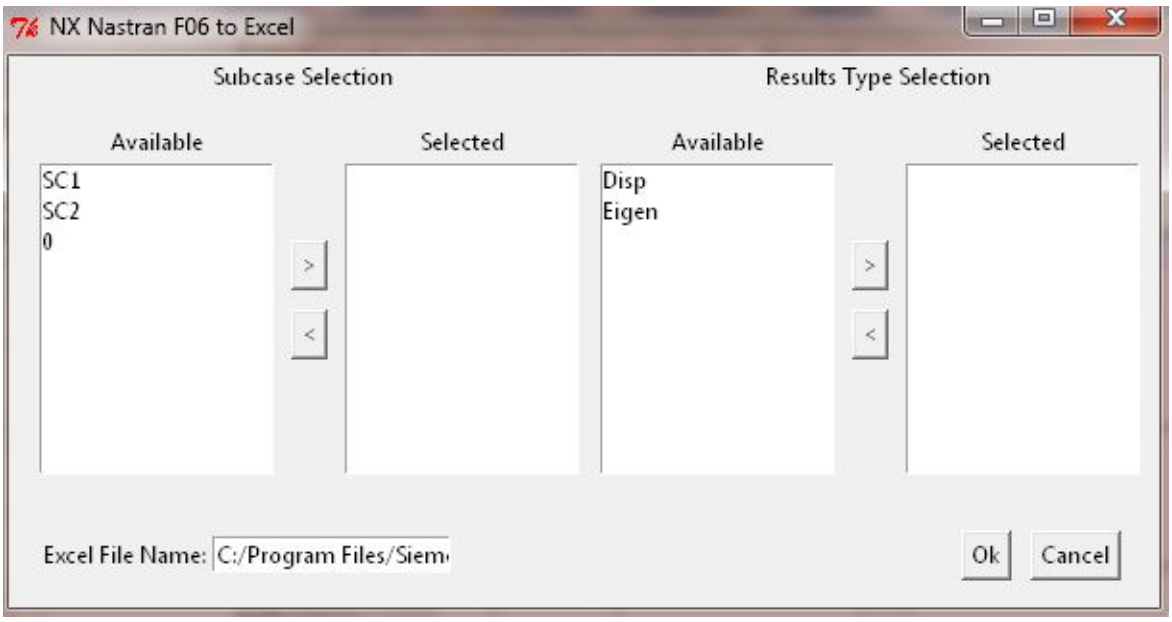
You can run the translator without the interface by removing the “-g” from the command line input. For example, if you enter

```
installation_directory/nxnr/util/toExcel.pl file.f06
```

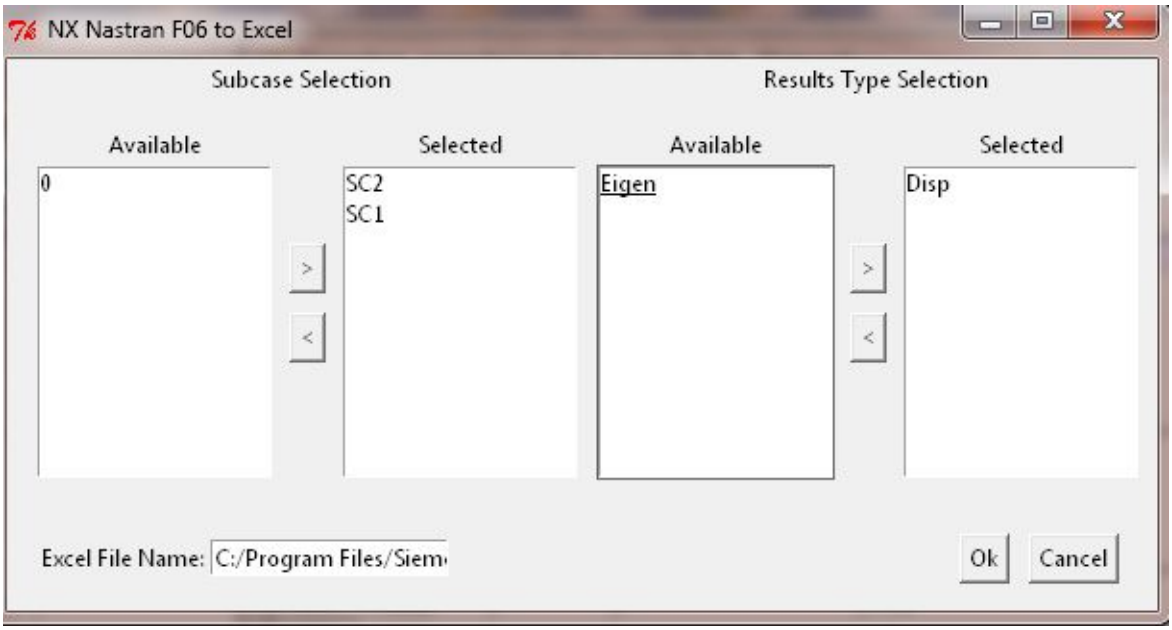
at the command line prompt, the translator will select and transfer all of the data in the .f06 file that is supported by the translator.

When you use the interface option, the output data types in the .f06 file which are supported by the translator are displayed in the interface for you to include or exclude in the data transfer.

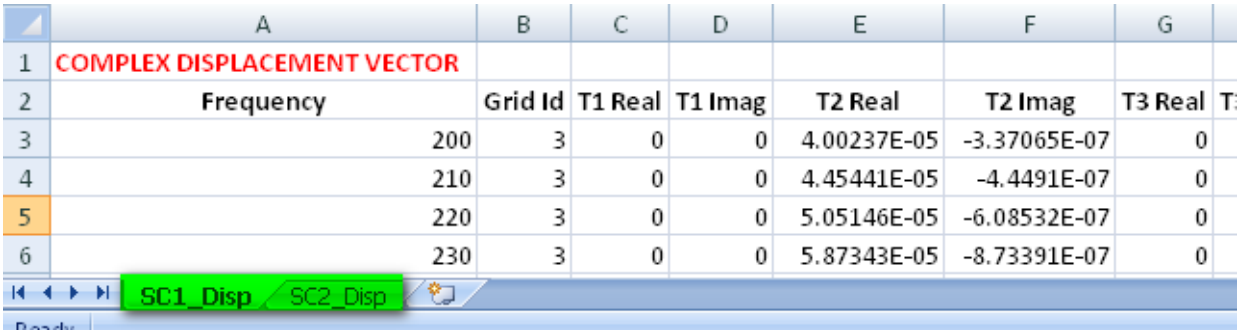
For example, an .f06 file includes displacement output for two subcases from a modal frequency response solution. The frequency-dependent displacement output is in SORT2 format. After you select the .f06 file, the translator displays the file contents in the interface.



You highlight any subcases and results which you would like to include in the data transfer, and then use the **>** button to move them into the **Selected** column. The subcase labeled “0” designates the global subcase. In the following interface image, subcases one and two have been selected along with the displacement result. Clicking **Ok** starts the data transfer.



After the data transfer is complete, Excel is automatically opened. The data for each subcase is written into separate Excel sheets as shown.



	A	B	C	D	E	F	G
1	COMPLEX DISPLACEMENT VECTOR						
2	Frequency	Grid Id	T1 Real	T1 Imag	T2 Real	T2 Imag	T3 Real
3	200	3	0	0	4.00237E-05	-3.37065E-07	0
4	210	3	0	0	4.45441E-05	-4.4491E-07	0
5	220	3	0	0	5.05146E-05	-6.08532E-07	0
6	230	3	0	0	5.87343E-05	-8.73391E-07	0

SC1_Disp SC2_Disp

You can optionally copy the translator into a new location. To do so, copy the following four files from the *util* folder in the NX Nastran installation hierarchy to a new location.

```
nxnExcel.pm
nxnExcelGui.pm
toExcel.pl
toExcel.xml
```

Note

The 11 character grid and element ID option selected with system cell 525 alters the .f06 file format. These .f06 files are not supported by the translator.

Output supported by the translator

- **Nodal output**
  - DISPLACEMENT
  - VELOCITY
  - ACCELERATION

- SPCFORCE

- **Element output**
  - STRESS
  - FORCE
  - STRAIN
- **Other output**
  - RCROSS
  - PANCON
  - MODCON
- **Output types**
  - Non-PSD and PSD
  - SORT1 and SORT2
  - Real and Complex
  - Centroid and Corner

Installing Perl and Packages

- **Installing Perl**

  A local installation of Perl is required to use the translator, along with some Perl packages. Perl can be downloaded from

  http://www.activestate.com/activeperl/downloads

- **Including Perl in your path**

  After you install Perl and the Perl Package Manager (PPM), you must include the location of the Perl executable in your path. For example, you can add

  ```
  perl_installation_location\Perl\bin
  ```

  to the environment variable

  ```
  Path
  ```

  To verify that Perl is in your path, enter the following option at the command prompt.

  ```
  Perl –v
  ```

  When you do so, the following message is displayed.

```
D:\>perl -v

This is perl 5, version 14, subversion 2 (v5.14.2) built for MSWin32-x86-mult
(with 1 registered patch, see perl -V for more detail)

Copyright 1987-2011, Larry Wall

Binary build 1402 [295342] provided by ActiveState http://www.ActiveState.com
Built Oct  7 2011 15:49:44

Perl may be copied only under the terms of either the Artistic License or the
GNU General Public License, which may be found in the Perl 5 source kit.

Complete documentation for Perl, including FAQ lists, should be found on
this system using "man perl" or "perldoc perl".  If you have access to the
Internet, point your browser at http://www.perl.org/, the Perl Home Page.
```

- **Installing Perl Packages**

  The Perl packages Excel-Writer-xlsx, Tkx, and XML-Parser must be installed.

  To open the Perl Package Manager, on the Windows taskbar, choose **Start→All Programs**, and double-click **Perl Package Manager**.

  In the Perl Package Manager, click **View all packages** .

  - In the filter field, type **excel**.

    Right-click Excel-Writer-XLSX in the package list and choose **Install Excel-Writer-XLSX** if it is not already installed.

  - In the filter field, type **tkx**.

    Right-click Tkx in the package list and choose **Install Tkx** if it is not already installed.

  - In the filter field, type **xml-parser**.

    Right-click XML-Parser in the package list and choose **Install Xml-Parser** if it is not already installed.

  - Select **Run marked Actions** under **File** to install packages.

7.3 ESTIMATE

You can use ESTIMATE to estimate the memory and disk requirements for NX Nastran jobs and make suggestions on improving the performance of these jobs. ESTIMATE reads the input data file and estimates the job's memory and disk requirements.

The ESTIMATE program is most accurate in predicting the requirements of static analyses that do not have excessive output requests. The memory requirements for normal modes analyses using the Lanczos method are reasonably accurate; however, the disk requirements are dependent upon the number of modes, this is a value that ESTIMATE cannot determine. Memory and disk requirements for other solutions are less accurate.

The basic format of the "estimate" command is

```
nxnr estimate input_file [keywords]
```

where input_file is the name of the data file. If the file type of the input data file is “.dat”, it may be omitted from the command line.

ESTIMATE processes keywords using the following precedence to resolve conflicts when keywords are duplicated (with 1 representing the highest precedence):

1. The Bulk Data file.
2. The command line.
3. The nastran INI and RC files (if “nastrc=yes” is specified).
4. *data-file-directory*/.estimaterc on Linux, or *data-file-directory*\estimate.rcf on Windows, where *data-file-directory* is the directory containing the input data file.
5. $HOME/.estimaterc on Linux, or %HOMEDRIVE%%HOMEPATH%\estimate.rcf file on Windows.
6. estimate.ini in the directory containing the ESTIMATE executable.

Please be aware that the Bulk Data file can only contain statements that are accepted by NX Nastran. The following keywords will be recognized by ESTIMATE when they appear in the Bulk Data file on NASTRAN statements:

```
buffpool, buffsize, real
```

Note

"buffsize=estimate" is NOT accepted on a NASTRAN statement.

The following Case Control statements will be recognized by ESTIMATE when they appear in the bulk data file:

```
adapt, method, mpc, sp
```

Note

If these statements appear multiple times, e.g., in subcases, only the first occurrence of each case control statement will be recognized.

Similarly, the nastran INI and RC files can only accept keywords that are accepted by the nastran command. The following nastran command keywords will be recognized by ESTIMATE when they appear in nastran RC files if and only if "nastrc=yes" is also set:

```
bpool, buffsize, memory, real, realdelta, smemory, version
```

The full set of ESTIMATE utility keywords can ONLY appear on the ESTIMATE command line or in the ESTIMATE RC files, e.g., ".estimaterc" on Linux and "estimate.rcf" on Windows.

Keywords

adapt	adapt=*number*	Default:	None

Selects an ADAPT set for adaptivity jobs if an ADAPT Case Control command is not present or multiple ADAPT Case Control commands are present in the data file. By default, ESTIMATE will choose the first ADAPT found.

bpool bpool=*value* Default: 37

Same as NX Nastran keyword, see “bpool”. This keyword cannot appear in an ESTIMATE RC file if “nastrc=yes” is specified.

buffsize buffsize=*number* Default: 8193

Same as NX Nastran keyword, see “buffsize”. This keyword cannot appear in an ESTIMATE RC file if “nastrc=yes” is specified.

dballco dballco=*value* Default: 1

Allows you to scale DBALL estimates. This scale factor is applied before the "dballmin" value, that provides a lower bound for DBALL estimates.

Example: `nxnr estimate example dballco=2`

This will double the DBALL disk estimate and then apply the "dballmin" lower bound.

Example: `nxnr estimate example dballco=0.5`

This will halve the DBALL disk estimate. An estimate less than than the lower bound specified by "dballmin" will be set to the lower bound.

dballmin dballmin=*value* Default: 1mb

Allows you to define the lower bound for all DBALL estimates. This bound is applied after the "dballco" value, that multiplies the actual estimate by a "conservatism" factor.

Example: `nxnr estimate example dballmin=2mb`

This will set the minimum DBALL disk estimate to 2 MB.

dskco dskco=*value* Default: 1

Allows you to define a factor to scale total disk estimates. This scale factor is applied before the "dskmin" value, that provides a lower bound for total disk estimates.

Example: `nxnr estimate example dskco=2`

This doubles the total disk estimate and then applies the "dskmin" lower bound.

Example: `nxnr estimate example dskco=0.5`

This will halve the total disk estimate. An estimate less than the lower bound specified by "dskmin" will be set to the lower bound.

dskmin

dskmin=*value* Default: 1mb

Allows you to define the lower bound for all total disk estimates. This bound is applied after the "dskco" value, that multiplies the actual estimate by a "conservatism" factor.

Example: `nxnr estimate example dskmin=2mb`

This will set the minimum total disk estimate to 2 MB.

enable

The “enable” keyword can be used to explicitly enable rules. This may be useful to enable a rule that was automatically suppressed when a value was assigned. For example, the following command will now calculate the estimated memory requirements for a job even though a value for memory was specified on the command line:

Example: `nxnr estimate example memory=5mb enable=10`

estimatedof

estimatedof=yes,no Default: No

Indicates if the number of degrees of freedom are to be estimated. By default, ESTIMATE will count the DOF. This process takes time, but it is generally more accurate. Specifying “estimatedof=no” will result in a less accurate, but faster, estimate of the DOF. The presence of any MESH entries in the Bulk Data will force “estimatedof=yes”.

memco

memco=*number* Default: 1.0

Allows you to specify a constant factor that is either more or less conservative than the default.

Example: `nxnr estimate example memco=2`

This setting will double the memory estimate.

memmin

memmin=value Default: 16mb

Allows you to define the lower bound for all memory estimates. This bound is applied after the "memco" value, that multiplies the actual estimate by a "conservatism" factor.

Example: `nxnr estimate example memmin=8mb`

This will set the minimum memory estimate to 8 MB.

memory

memory=*size* Default: estimate

Same as NX Nastran keyword, see "memory" . This keyword cannot appear in an ESTIMATE RC file if "nastrc=yes" is specified.

method method=*number* Default: None

Selects a METHOD for dynamics jobs if a METHOD Case Control command is not present or multiple METHOD Case Control commands are present in the data file. By default, ESTIMATE will choose the first METHOD found.

mode mode=*keyword* Default: suggest

Selects the program operating mode. Specifying "mode=estimate" will result in memory and disk estimates only. Specifying "mode=suggest", the default, will estimate memory and disk requirements for the current job configuration, suggest modifications to improve the performance, and provide estimates for the memory and disk requirements of the suggested configuration. Specifying "mode=modify" does all that "mode=suggest" does plus actually make the suggested changes to your data file. See "out" to specify the new data file's name and information on organizing your input file.

Note

If "mode=modify" is specified, and ESTIMATE detects errors in the input file or encounters valid Bulk Data that is not understood by ESTIMATE, the program will revert to "mode=suggest".

Example: `nxnr estimate example mode=estimate`

The memory and disk requirements for the current job are displayed.

Example: `nxnr estimate example`

The memory and disk requirements for the current job, suggestions for improving performance, and memory and disk requirements for the suggested configuration are displayed.

Example: `nxnr estimate example mode=modify`

The memory and disk requirements for the current job, suggestions for improving performance, and estimates of memory and disk requirements for the suggested configuration are displayed. If, and only if, modifications to "example.dat" are suggested, the original input file is versioned (given indices) and the revised data file is written to "example.dat".

mpc mpc=*number* Default: None

Selects an MPC if an MPC Case Control command is not present or multiple MPC Case Control commands are present in the data file. By default, ESTIMATE will choose the first MPC found.

nastrc nastrc=yes,no Default: Yes

The “nastrc” keyword allows you to select the type of RC file processing invoked by the ESTIMATE utility. Setting “nastrc=yes”, the default, will process the standard NX Nastran RC files before the standard ESTIMATE RC files, i.e., $HOME/.estimaterc and “*data-file-directory*/.estimaterc” on Linux, and %HOMEDRIVE%%HOMEPATH\estimate.rcf and “*data-file-directory*/estimate.rcf” on Windows, are processed. Setting “nastrc=no” will only process the standard ESTIMATE RC files.

out out=*pathname* Default: input filename

Specifies the name of the output file if “mode=modify” is specified and modifications of the data file are actually required. By default, the original file is versioned (given indices) and the revised data file is written to the original input file’s name. See “Using Filenames and Logical Symbols”

Example:

```
nxnr estimate example mode=modify
```

If modifications to “example.dat” are suggested, the original input file is versioned (given indices) and the revised data file is written to “example.dat”.

Example:

```
nxnr estimate example mode=modify \
out=modified
```

The revised data file is written to “modified”.

Note

To minimize the amount of data duplicated between the original input file and the modified file, the Bulk Data that is not subject to modification by ESTIMATE (i.e., all Bulk Data except PARAM and EIGRL entries) should be placed in an INCLUDE file.

An example of the recommended input file organization is:

```
NASTRAN statements
FMS statements
Executive
CEND Case Control
BEGIN BULK
PARAM,...
$
EIGRL,...
$
INCLUDE file.bulk
$ ENDDATA
```

pause pause=*keyword* Default: No

Pause ESTIMATE before exiting to wait for the “Enter” or “Return” key to be pressed. This can be useful when ESTIMATE is embedded within another program. The values are “fatal”, “information”, “warning”, “yes”, and “no”. Setting “pause=yes” will unconditionally wait; “pause=fatal” will only wait if a fatal message has been issued by ESTIMATE; “pause=information” and “pause=warning” will similarly wait only if an information or warning message has been issued. The default is “pause=no”, i.e., do not wait when ESTIMATE ends.

real real=*value* Default: See text.

Same as NX Nastran keyword, see “real” . This keyword cannot appear in an ESTIMATE RC file if “nastrc=yes” is specified.

realdelta realdelta=*value* Default: See text.

Same as NX Nastran keyword, see “realdelta”. This keyword cannot appear in an ESTIMATE RC file if “nastrc=yes” is specified.

report report=*keyword* Default: Normal

Specifies the program’s report format. The “report=normal” format is intended to be read by you. The “report=keyword” format is intended to be read by a program.

scr300co scr300co=*value* Default: 1

Allows you to define a factor to scale SCR300 estimates. This scale factor is applied before the "scr300min" value, that provides a lower bound for SCR300 estimates.

Example: `nxnr estimate example scr300co=2`

This will double the SCR300 disk estimate and then apply the "scr300min" lower bound.

Example: `nxnr estimate example scr300co=0.5`

This will halve the SCR300 disk estimate. An estimate less than the lower bound specified by "scr300min" will be set to the lower bound.

scr300min scr300min=*value* Default: 1mb

Allows you to define the lower bound for all SCR300 estimates. This bound is applied after the "scr300co" value, that multiplies the actual estimate by a "conservatism" factor.

Example: `nxnr estimate example scr300min=2mb`

This will set the minimum SCR300 disk estimate to 2 MB.

scratchco scratchco=*value* Default: 1

Allows the user to define a factor to scale SCRATCH estimates. This scale factor is applied before the "scratchmin" value, that provides a lower bound for SCRATCH estimates.

Example: `nxnr estimate example scratchco=2`

This will double the SCRATCH disk estimate and then apply the "scratchmin" lower bound.

Example: `nxnr estimate example scratchco=0.5`

This will halve the SCRATCH disk estimate. An estimate less than the lower bound specified by "scratchmin" will be set to the lower bound.

scratchmin scratchmin=*value* Default: 1mb

Allows you to define the lower bound for all SCRATCH estimates. This bound is applied after the "scratchco" value, that multiplies the actual estimate by a "conservatism" factor.

Example: `nxnr estimate example scratchmin=2mb`

This will set the minimum SCRATCH disk estimate to 2 MB.

smemory smemory=*size* Default: 100 (all others)

Same as NX Nastran keyword, see “smemory”. This keyword cannot appear in an ESTIMATE RC file if “nastrc=yes” is specified.

spc spc=*number* Default: None

Selects an SPC if an SPC Case Control command is not present or multiple SPC Case Control commands are present in the data file. By default, ESTIMATE will choose the first SPC found.

suppress suppress=*list* Default: None

Specifies rules that are to be suppressed when “mode=suggest” or “mode=modify” is specified. See “Rules” for the list of rules. If no value is specified, i.e., “suppress=”, then any rules previously suppressed are enabled. Multiple rules can be suppressed by using the keyword multiple times or by specifying a comma-separated list.

Example: `nxnr estimate example suppress=1`

Suppress rule 1, the rule controlling BUFFSIZE.

Examples:

```
nxnr  estimate example suppress=1,6

nxnr  estimate example suppress=1 suppress=6

nxnr  estimate example suppress=2 suppress= \

suppress=1,6
```

Suppress rules 1 and 6.

verbose verbose=yes,no Default: No

Specifies the amount of information to be displayed. Specifying "verbose=yes" will generate a much larger amount of output. The additional information includes a more detailed summary of the input file, the parameters used in estimating the memory and disk requirements, and the estimates for the original file, even when "mode=suggest" or "mode=modify" is specified.

version version=*string* Default: 1

Specifies the version for which the estimates are targeted. The version will affect the estimated memory requirements and the actions of various rules, see "Rules" . This keyword cannot appear in an ESTIMATE RC file if "nastrc=yes" is specified.

wordsize wordsize=*number* Default: = 32 using the IL-64 executables

= 64 using ILP-64 executables

Specifies the word size of the estimate's target computer. By default, ESTIMATE's calculations will be appropriate the current computer. This keyword may be used to specify estimates for a computer with a different word size. A comma-separated list of values may be specified when estimates and suggestions for multiple machines are desired. If "mode=modify" was specified, the modification are based on the last word size specified.

Rules

ESTIMATE has a fixed rule base that it uses to make suggestions for improvement. You can suppress any of the rules with the "suppress" keyword. The current rules are:

1. Set recommended BUFFSIZE.

BUFFSIZE=8193	*DOF* ≤100000
BUFFSIZE=16385	100000 < *DOF* ≤ 400000
BUFFSIZE=32769	*DOF* > 400000

2. Use default BPOOL.

BPOOL=37	*wordsize* = 32
BPOOL=20	*wordsize* = 64; *version* < 70.5
BPOOL=27	*wordsize* = 64; *version* ≥ 70.5

3. Suppress symmetric decomposition if not enough memory for sparse.

SYSTEM(166)=0

4. Make all open core available to modules.

   Delete HICORE.

5. Select the sparse solver.

Delete SPARSE Delete USPARE	*density* ≤12.0
SPARSE=1 USPARSE=0	*density* >12.0

6. Force default rank size.

   Delete SYSTEM(198)

   Delete SYSTEM(205)

7. Do not sequence.

   PARAM,NEWSEQ,-1 *version* < 69.0

8. Use default Lanczos parameters.

   EIGRL,...,V1=""

   EIGRL,...,MAXSET=15

9. Use default SMEMORY.

   INIT SCRATCH (MEM=100) *wordsize* = 32

   INIT SCRATCH (MEM=0) *wordsize* = 64

10. Use estimated memory size.

    memory=*estimated-memory*

11. Use default RAM.

    INIT MASTER (RAM=30000)

12. Real.

    Delete REAL.

13. Do not use Supermodule.

    Delete PARAM,SM,YES.

14. Do not use Parallel Lanczos.

    Delete NUMSEG.

Examples

The ESTIMATE program can be used in several ways. In the default mode, ESTIMATE makes suggestions on improving the performance of NX Nastran and estimates the resource requirements of the job assuming the suggested parameters.

```
nxnr estimate example
```

To get an estimate of the job using the current parameters, use the command:

```
nxnr estimate example mode=estimate other_estimate_keywords
```

To have a new input file generated with the suggested changes, use the command:

```
nxnr estimate example mode=modify other_estimate_keywords
```

To run NX Nastran with the memory estimated by ESTIMATE, use:

```
nxnr nastran example memory=estimate other_nastran_keywords
```

7.4 F04REPRT

The F04REPRT utility is a Perl script that will summarize and/or compare .f04 files. The utility can determine the CPU time consumed by various NX Nastran modules, i.e., as a DIAG 49 replacement, or compare the relative performance of one or more jobs under various configurations.

Note

You must have Perl installed on your system to use this utility.

The basic format of the F04REPRT command is

```
nxnr f04reprt.pl -s [options] pathname [pathname ...]
```

or

```
nxnr f04reprt.pl -d [options] old1 new1 [oldn newn...]
```

where “-s” selects the summary mode, “-d” selects the comparison mode, *options* are zero or more of the options listed below, pathname is a pathname, and *oldi* and *newi* are pathnames. If a *pathname* is a directory, all .f04 files in the directory are summarized/compared.

Note

Alternatively, you can run F04REPRT with the command

```
perl install-dir/nxnr/util/f04reprt.pl arguments ...
```

on Linux, or

```
perl install-dir\nxnr\util\f04reprt.pl arguments ...
```

on Windows if perl is in your PATH. Linux users can also use the command

```
install-dir/nxnr/util/f04reprt.pl arguments ...
```

if your Perl executable is /usr/local/bin/perl, or the “shbang” line was updated to the appropriate path.

Running F04REPRT without any arguments will display a help message explaining the utility's options.

Options

-c -c Default: No

Indicates module times are to be accumulated in a single entry, rather than separate entries for each module occurrence

-d -d Default: None

Requests a comparison (difference) between each pair of oldi and newi pathnames specified on the command line. If only one pair of pathnames are specified, the "-d" is optional.

-f -f *c* Default: Space

Specifies a field separator character to separate field in the comparison report. This character may be inclosed in either single or double quotes to protect it from the command shell.

-m -m *number* Default: 0.05

Specifies the minimum CPU time threshold for comparisons or summaries. CPU times less than this threshold will be ignored.

-o -o *file-type* Default: None

Specifies an output file-type. If specified, each comparison or summary report will be written to a separate file in the current working directory with the name basename.ext where basename is the base name of the pathname or oldi.

If not specified, output will be written to stdout with each report separated by a form feed "Ctrl-L" character.

-r -r *number* Default: 5

Specifies the delta percentage used for "FASTER" and "SLOWER" comments in comparison (-d) output.

Any old versus new comparisons that exceed this delta from 100%,eg., (*delta* < 95%) or (*delta* > 105%) , will print the appropriate comment.

-s -s Default: None

Requests a summary report for each pathname specified on the command line. If only one pathname is specified, the "-s" is optional.

-x -x *file-type* Default: f04

Specifies an alternate input file type.

Examples

```
nxnr f04reprt.pl example
```

If "./example" on Linux, or ".\example" on Windows, is a subdirectory of the current directory, F04REPRT will write a summary report to stdout for every .f04 file in the directory. Otherwise, if "./example.f04" on Linux, or ".\example.f04", on Windows is a file, a summary report of the one file is written to stdout.

```
nxnr f04reprt.pl old new
```

If "old" and "new" are subdirectories of the current working directory, F04REPRT will generate lists of the .f04 files in each directory. Comparisons will be made between each pair of files with the same name in the two directories. Non-.f04 files and unpaired .f04 files, i.e., .f04 files that exist in either

"old" or "new" but not both, will be ignored. Otherwise, if "old.f04" and "new.f04" are files, then a comparison of these two files will be displayed.

7.5 EDSACT

EDSACT may be used to generate usage reports from the accounting files generated by NX Nastran when the "acct=yes" keyword is used. The basic format of the "edsact" command is

```
nxnr edsact [keywords] acc-file [acc-file ...]
```

where *acc-file* are the names of the accounting file(s) to be summarized.

Note

The keywords only affect files listed after the keyword.

Keywords

perfile perfile=yes,no Default: No

Specifies the summary is to be printed on a per file basis. If "perfile=yes" is specified, a summary of each file will be individually printed. By default, the summary will include all files.

sortby sortby=*keyword* Default: Name

Sort the report as specified by the keyword. The keywords are:

Keyword	Sort Order
no	Do not sort report; report is ordered as found in data file.
count	Sort by count column.
name	Sort by name column.
time	Sort by time column.

summary summary=*keyword* Default: None

Selects the type of summary. If “summary=none” is specified, the total CPU for all entries will be displayed. Otherwise, one of the following summary types may be selected:

Keyword	Type of Summary
acdata	By acdata
acid	By account ID (acid)
date	By execution date
jid	By job name
product	By product name
sol	By SOL
user	By user name
version	By product name and version

Examples

All of the following examples assume your current working directory is the NX Nastran accounting directory, i.e., *install_dir*/acct on Linux and *install_dir*\acct on Windows.

To summarize accounting data across all files:

```
nxnr edsact file1 file2

file1 file2:

Total:  cpu-sec count
```

where *filei* are the filenames, *cpu-sec* is the total CPU seconds across all files, and count is the number of entries accumulated across all files.

To summarize accounting data from individual files:

```
 nxnr edsact perfile=yes file1 file2
file1:
  Total: cpu-sec count
file2:
  Total: cpu-sec count
```

where file, is the name of each file, cpu-sec is the total number of CPU seconds, and count is the number of entries in each file.

To summarize accounting data in individual files by user:

```
nxnr edsact summary=user perfile=yes file1 file2
file1:
  user1: cpu-sec1 count1
  user2: cpu-sec2 count2
   ...
   Total: cpu-sec count
file2:
  user1: cpu-sec1 count1
  user2: cpu-sec2 count2
```

```
...
 Total: cpu-sec count
```

where *filei* are the filenames of each file, *useri* are the names, *cpu-seci* are the total CPU seconds for each user, *counti* are the number of entries accumulated for each user, *cpu-sec* is the number of total CPU seconds, and count is the number of entries in each file.

Accounting File Format

A separate file is created for each month of each year and is named

install_dir/acct/nxn*yymm*.acc

on Linux and

install_dir\acct\nxn*yymm*.acc

on Windows where *yy* are the last two digits of the year and *mm* is the month (01 to 12). Each month's file is independent of every other file.

The accounting file begins with three header records followed by detail records, one detail record for each NX Nastran job run during the given month and year. Comments, indicated by a hash mark "#" as the first character of the line, may be placed anywhere in the file after the header records.

Detail records (any non-comment line after the third line) include the following data:

1. The day the job was started (i.e., Sun., Mon., Tue., Wed., Thu., Fri., or Sat.).
2. The month the job was started (i.e., Jan., Feb., Mar., Apr., May, Jun., Jul., Aug., Sep., Oct., Nov., or Dec.).
3. The date of the month the job was started (i.e., 01 through 31).
4. The time the job was started (i.e., hh:mm:ss, where hh is 00 through 23, mm is 00 through 59, and ss is 00 through 59).
5. The time zone (i.e., the "TZ" environment variable).
6. The year the job was started (four digits).
7. The name of the user running the job.
8. The job's output filename.
9. The analysis application.
10. The version of the application.
11. The SOL used by the job (e.g., 101, SESTATICS).
12. The total CPU time, in seconds, of the job (from the .f04 file).
13. The cumulative CPU time, in seconds, of all detail records up to and including this record.
14. The cumulative CPU time, in minutes, of all detail records up to and including this record.

15. The account ID as specified by the nastran command's "acid" keyword.

16. The account data as specified by the nastran command's "acdata" keyword.

Note

The cumulative times (fields 13 and 14) are for historical purposes only. These values are ignored.

7.6 HEATCONV

HEATCONV may be used to reformat an existing heat-transfer Bulk Data file used in MSC.Nastran®(MSC.Nastran is a registered trademark of MSC.Software Corporation) prior to Version 68 into a format compatible with Version 68 or later. The operations performed by this program are described in the MSC.Nastran®Release Notes for Version 68 (MSC.Nastran is a registered trademark of MSC.Software Corporation). The basic format of the "heatconv" command is

```
nxnr heatconv input_file [keywords]
```

where input_file is the name of the heat-transfer data file. If the file type of the old data file is ".dat", it may be omitted from the command line.

Keywords

output output=*pathname* Default: input_file

This option specifies the name of the reformatted data file. By default, the old output file is renamed by appending the file type ".old"; the new file is the original name of the input file. If an output file is specified using this option, the original input filename is unchanged.

Examples

To execute the program, enter the following command:

```
nxnr heatconv example
```

The Version 68-compatible output is written to

```
example.dat
```

The original data file is renamed to example.dat.old.

7.7 MSGCMP

MSGCMP compiles a text message file and generates a binary message catalog. The basic format of the command is

```
nxnr msgcmp text_file [message_catalog]
```

where text_file is the name of an existing text message file or is "-" to read from stdin, and *message_catalog* is the optional name of the message catalog that will be written. The type of the

text file must be ".txt". If a message catalog is not named, the message catalog will be written in the local directory as "*text_file.msg*". The message catalog can be tested using the "msgcat" keyword.

The utility can also regenerate a text file from an existing message catalog using the command

```
nxnr msgcmp message_catalog.msg [text_file]
```

where message_catalog.msg is the name of an existing message catalog and text_file is the optional name of a text file that will be written. The type of the message catalog must be ".msg" and must be entered on the command line. If a text file is not named, the text file is written to stdout.

The text source file for the standard message catalog is

```
install_dir/nxnr/util/analysis.txt
```

on Linux and

```
install_dir\nxnr\util\analysis.txt
```

on Windows. The standard message catalog is

```
install_dir/nxnr/arch/analysis.msg
```

on Linux and

```
install_dir\nxnr\arch\analysis.msg
```

on Windows.

Examples

The following command will compile the message catalog from a text file named "myfile.txt"

```
nxnr msgcmp myfile
```

The message catalog will be named "myfile.msg". This catalog may be used with the nastran command

```
nxnr nastran myjob msgcat=myfile.msg other_nastran_keywords
```

Note

Message catalogs are machine dependent. "Binary File Compatibility" identifies the systems that are binary compatible; binary compatible systems can use multiple copies of the same message file.

7.8 NEUTRL

NEUTRL converts a binary-format plot file into a neutral-format plot file. The basic format of the "neutrl" command is

```
nxnr neutrl binary_plot_file [keywords]
```

where *binary_plot_file* is the name of a binary plot file. If the file type of the plot file is ".plt", it may be omitted from the command line.

Keywords

dump	dump=yes,no	Default:	no

This option enables a raw print of each plot command to be made before it is processed. This print is used for debugging purposes only.

output	output=*pathname*	Default:	*binary_plot_file*.neu

This option specifies the name of the neutral-format file. If “out=—” is specified, the neutral plot file is written to stdout. By default, the output file is the name of the input file with the new type “.neu”.

verbose	verbose=yes,no	Default:	yes *Processing messages written to stdout*. no *Processing messages not written to stdout.*

Examples

To execute the program, enter the following command:

```
nxnr neutrl example1
```

The name of the output file is

```
example1.neu
```

7.9 OPTCONV

OPTCONV may be used to reformat an existing optimization Bulk Data file used in MSC.Nastran® (MSC.Nastran is a registered trademark of MSC.Software Corporation) prior to Version 68 into a format compatible with Version 68 or later. The operations performed by this program are described in the M*SC.Nastran® Release Notes for Version 68* (MSC.Nastran is a registered trademark of MSC.Software Corporation). The basic format of the “optconv” command is

```
nxnr optconv input_file [keywords]
```

where *input_file* is the name of the dynamic-optimization data file. If the file type of the old data file is “.dat”, it may be omitted from the command line.

Keywords

output	output=*pathname*	Default:	input-file

This option specifies the name of the reformatted data file. By default, the old output file is renamed by appending the file type ".old"; the new file is the original name of the input file. If an output file is specified using this option, the original input filename is unchanged.

Examples

To execute the program, enter the following command:

```
nxnr optconv example
```

The Version 68-compatible output is written to

```
example.dat
```

The original data is renamed to example.dat.old.

7.10 PLOTPS

PLOTPS reads plotting commands from a single NX Nastran binary- or neutral-format plot file and produces a file that can be printed on a PostScript device. The basic format of the "plotps" command is

```
nxnr plotps input_plot_file [keywords]
```

where input_plot_file is the name of the plot file generated by NX Nastran or NEUTRL. A neutral-format plot file can be read from stdin by specifying "-" as the filename. The plot file type ".plt" does not have to be specified on the command line.

Keywords

begin	begin=*number*	Default:	1
end	end=*number*	Default:	999999
	Plots a selected range of plot frames.		
color	color=yes,no	Default:	No
	Enables or disables color pens. Setting "color=no", the default, will assign a solid line to pen 1 and various dashed lines to pens 2, 3, and 4. Setting "color=yes" will assign black to pen 1, red to pen 2, green to pen 3, and blue to pen 4. All text and axes will always be written with a solid black pen.		
cscale	cscale=*number*	Default:	1.0
	Specifies a scale factor for all characters and special symbols on the plot. By default, characters and special symbols are 9 points (about 0.125 inch). The scale value, if specified, is also applied to characters and special symbols. The "cscale" value is critical to the correct imaging of the plot if "optimizestrings=yes" was specified. In general, you must specify the same "cscale" value as was specified in the original NX Nastran job that generated the PLT file.		
dump	dump=yes,no	Default:	No
	Enables a raw print of each plot command before it is processed. This print is used for debugging purposes only.		
format	format=*keyword*	Default:	Binary

Specifies the input file format. If the file type of the input file is ".neu" or the plot file is read from stdin, then "format=neutral is assumed.

height height=*number* Default: 10.0 inches

Specifies the printable page height. The actual page is assumed to be 1 inch larger.

optimizestrings optimizestrings=yes,no Default: Yes

Indicates the string optimization feature is to be enabled. This feature can result in a substantial reduction in plot file size, printer memory requirements, and print speed.

If "optimizestrings=no" is set, PLOTPS will draw each character individually, at the expense of PS file size and the memory and time needed by your PostScript printer to image the file.

output output=*pathname* Default: *plot-file*.ps

Specifies the name of the PostScript output file. If a neutral-format plot file is read from stdin, the default output filename is "plotps.ps". If "out=—" is specified, the PostScript output is written to stdout. By default, the output file is named the name of the input file with the new type ".ps".

rotate rotate=*keyword* Default: Automatic

Controls the orientation of the generated image. If "rotate=automatic" is specified, the program orients the image so that the long direction of the image is aligned with the long direction of the page. If "rotate=no" is specified, the image is generated with the horizontal axis aligned with the bottom edge of the page. If "rotate=yes" is specified, the image is generated with the horizontal axis aligned with the right edge of the page.

scale scale=*number* Default: 1.0

Specifies a scale factor for all elements of the plot.

Note

The program will not attempt to print a multipage image if this option is used to enlarge the image beyond the size of the available page.

verbose verbose=yes,no Default: Yes *Output is a disk file*; No *Output is stdout*

Specifies whether processing messages are to be written.

width width=*number* Default: 7.5 inches

Specifies the printable page width. The actual page is assumed to be 1 inch larger.

Examples

To translate a binary-format plot file named example1.plt into PostScript, use

```
nxnr plotps example1
```

The name of the output file is

```
example1.ps
```

To translate a neutral-format plot file named example2.neu into PostScript, use

```
nxnr plotps example2.neu
```

The name of the output file is

```
example2.ps
```

Using the String Optimization Feature

When the string optimization feature functions correctly, you can realize a substantial reduction in the size of the PostScript file and a commensurate reduction in the memory and time needed by your PostScript printer to image the file. However, there are some cases where the feature does not function correctly, and generates an incorrect plot image.

The “cscale” value used in the NX Nastran job that generated the PLT file is critical to the correct operation of the “optimizestrings” feature. In general, you need to specify the same value in the PLOTPS run. There are some cases, however, where the value should be left at the default, i.e., 1.0. You can determine this by imaging and printing the first frame of the PLT file with the following two commands:

```
nxnr plotps plt-file end=1 out=value.ps cscale=cscale-value

nxnr plotps plt-file end=1 out=default.ps
```

where *plt-file* is the NX Nastran PLT file and *cscale-value* is the CSCALE value used in the NX Nastran job that generated the file. A visual comparison of the two PostScript images will identify the correct setting. In general, it will be the first command, i.e., the one that set the CSCALE value to the NX Nastran job’s value.

A summary of PostScript file sizes and “cscale” values is presented below for several TPL files:

File name	PLT File Size	PS File Size			“cscale” Value	
		V70	opt=no	opt=yes	NX Nastran	PLOTPS
d10112r	102272	137889	53129	22471	1.8	1.0
hd15901	57152	68644	27613	14605	1.8	1.8
pt1031	81216	100844	39345	14101	1.5	1.5
v14501q	15040	24343	10123	6767	1.2	1.2

Of these files, only “d10112r” used the default “cscale” value to image correctly.

7.11 RCOUT2

RCOUT2 is used to convert a neutral-format OUTPUT2 file generated by NX Nastran into a binary-format OUTPUT2 file. Since NX Nastran can read and write binary-format and neutral-format OUTPUT2 files, this utility is generally used to construct a binary OUTPUT2 file for a third-party program that can only read a binary OUTPUT2 file. The basic format of the “rcout2” command is

```
nxnr rcout2 neutral_output2_file [keywords]
```

where *neutral_output2*_file is the name of the neutral-format OUTPUT2 file. If the file type of the OUTPUT2 file is “.on2", it may be omitted from the command line.

Keywords

output	output=*pathname*	Default: *neutral_file*.op2
	This option specifies the name of the binary OUTPUT2 file. By default, the output file is the name of the input file with the new type ".op2".	

Examples

To execute the program, enter the following command:

```
nxnr rcout2 example
```

The name of the output file is

```
example.op2
```

7.12 RECEIVE

RECEIVE converts a neutral results database file (NDB) into a binary results database file (XDB). The basic format of the "receive" command is

```
nxnr receive neutral_xdb_file [keywords]
```

where neutral_xdb_file is the name of the NDB file. If "-" is specified as the neutral format database file, the file is read from stdin. If the file type of the NDB file is ".ndb", it may be omitted from the command line.

Keywords

output	output=*pathname*	Default:	*neutrl_xdb_file.xdb*
	This option specifies the name of the binary results database file. By default, the output file is the name of the input file with the new type ".xdb". If the neutral format database file was read from stdin, the default output filename is "receive.xdb". A binary XDB file cannot be written to stdout.		
verbose	verbose=yes,no	Default:	Yes *Output is a disk file* No *Output is stdout*.
	This option specifies whether processing messages are to be written.		

Examples

To execute the program, enter the following command:

```
nxnr receive example
```

The name of the output file is

```
example.xdb
```

On Linux systems, an XDB file can be transferred directly from a remote system with the following command

```
$ rsh node nxnr trans binary_xdb_file out=- \
| nxnr receive - out=binary_xdb_file
```

See the rsh(1) man page for further information.

7.13 TRANS

A results database file (XDB) may be exchanged between computer systems that have binary file compatibility as displayed in Table 7-2. Otherwise, the TRANS utility is required. TRANS converts an XDB file that is generated by NX Nastran to an equivalent character file that can be sent across a network to another computer. RECEIVE converts the character file back into an XDB file for postprocessing.

Binary File Compatibility

The following table lists the compatibility of binary files between various computer systems supported. Note that not all of these combinations have been tested. Please report any compatibility problems encountered

NX Nastran	Architecture			
	IEEE	Byte Order	Word Size	Intel
Linux and Windows	Yes	Little	32	Copy

Note

- Copy indicates that XDB files can be transferred between the systems without using TRANS and RECEIVE.

The first column on the left of the table lists various platforms that run NX Nastran. The second and third columns list basic architectural features of the computer, specifically whether the computer conforms to ANSI/IEEE Standard 754-1985 (the *IEEE Standard for Binary Floating-Point Arithmetic*) and byte ordering (big endian or little endian) used by the computer. The remaining columns list postprocessor platforms.

Running TRANS

TRANS converts a binary results database file (XDB) into a neutral results database file (NDB) that may be copied to any other computer. The basic format of the “trans” command is

```
nxnr trans binary_xdb_file [keywords]
```

where *binary_xdb_file* is the name of the XDB file. An XDB file cannot be read from stdin. If the file type of the XDB file is “.xdb”, it may be omitted from the command line.

Keywords

alphabet alphabet=*number* Default: 64

Choose the 48- or 64-character conversion table.

output output=*pathname* Default: *binary_xdb_file.ndb*

This option specifies the name of the neutral format database file. If “out=-” is specified, the neutral-format database file will be written to stdout. By default, the output file name is the input file name with the new type “.ndb”.

verbose verbose=yes,no Default: Yes: *Output is a disk file*
No: *Output is stdout*.

This option specifies whether processing messages are to be written.

Examples

To execute the program, enter the following command:

```
nxnr trans example
```

The name of the output file is

```
example.ndb
```

Input example:

```
nxnr trans binary_xdb_file out=- \
rsh node [-l user] nxnr receive - out=binary_xdb_file
```

See the remsh(1) or rsh(1) man pages for further information.

7.14 Building the Utilities Delivered in Source Form

Several of the utilities (i.e., PLOTPS, NEUTRL, RCOUT2, and EDSACT) are delivered in source and executable form. The source code allows these utilities to be customized or built for other platforms. A script and makefile are provided to build and install these utilities. The script determines the architecture of current platform and invokes the make utility to perform the actual compilation, link, and installation.

The utility program source files are located in

```
install_dir/nxnr/util
```

on Linux and

```
install_dir\nxnr\util
```

on Windows. This directory is an optional component of the NX Nastran installation. This directory includes the following files:

Table 7-2. Utility Program Source Files

File	Description
util	Script to Build Source Utility Programs.
ld.f	Source for RCOUT2 Utility Routines.
libfeds.F	Source for FORTRAN Utility Library Routines.
makefile	Makefile to Build Source Utility Programs.
mattst.F	Source for Sample OUTPUT2 File Reader MATTST (see “Building and Using MATTST”).
edsact.c	Source for NX Nastran Accounting Programs.
neutrl.F	Source for NEUTRL Utility.
ngtarg.F	Source for Command Line Utilities.

plotps.F	Source for PLOTPS Utility.
rcout2.F	Source for RCOUT2 Utility.
tabtst.F	Source for Sample OUTPUT4 File Reader TABTST (see “Building and Using TABTST”).

Three steps are required to build and install the source utilities. Make sure that you are in the utility program source directory, i.e., *install_dir*/nxn*r*/util on Linux and *install_dir*\nxn*r*\util on Windows.

1. The first step compiles and links all of the source utility programs. Enter the command

   ```
   nxnr util build
   ```

   If only one utility is to be built, use the name of the utility (i.e., “edsact,” “neutrl,” “plotps,” or “rcout2") instead of “build”. For example,

   ```
   nxnr util plotps
   ```

   will only build the PLOTPS utility.

2. After the programs are generated in the current directory, you can install the executable programs into the architecture directory for your computer (i.e., *install_dir*/nxn*r*/*arch* on Linux and *install_dir*\nxn*r**arch* on Windows). Enter the command

   ```
   nxnr util install
   ```

3. The third step deletes all object files and temporary files created by the “make” process. Enter the command

   ```
   nxnr util clean
   ```

The building and installation process can be repeated if you want to build the utilities for other computer architectures at your site.

To build the utilities on another computer that does not have NX Nastran installed, copy the complete utilities directory to the other computer. Since the nxn*r* command will not be available, you must run the util script directly. Before you do, however, set the environment variable *NXN_ARCH* to the name of a supported architecture as shown in Table 3-1. The “install” option cannot be used.

Chapter 8: Building and Using the Sample Programs

8.1 Overview

This chapter describes how to build and use the various NX Nastran sample programs. The sample programs are grouped by function as follows:

Program	Function
BEAMSERV	Implements user-defined bar and beam elements for NX Nastran.
DDLPPRT	Reads and displays XDB results database files. These sample programs are part of NX Nastran Access and demonstrate how to use the database library routines.
DDLQRY	
DEMO1	
DEMO2	
SMPLR	
MATTST	Reads and displays OUTPUT4 files.
TABST	

Descriptions on building and using the sample programs follow in alphabetical order.

8.2 Building and Using BEAMSERV

BEAMSERV implements a user-defined beam element for NX Nastran.

Note

The sample beam server source code is only provided as a simple example illustrating basic concepts. It is not intended to be a complete or usable program.

Unlike the other sample programs, a beam server is not a stand alone program that runs from the command line. Instead, the beam server is started and controlled by NX Nastran. In the current implementation, communications between NX Nastran and the beam server are accomplished through pipes, with NX Nastran reading and writing BEAMSERV's stdout and stdin units, respectively.

Note

1. The NX Nastran job invoking the beam server and the beam server itself must run on the same computer.

2. Your program may not read from stdin (FORTRAN logical unit 5) nor write to stdout (FORTRAN logical unit 6).

3. The beam server cannot write to the .f06, .f04, or .log files of the NX Nastran job that started the beam server.

4. Debugging must be accomplished by writing to a disk file, or connecting to the running beam server executable with a debugger (this may not be available on all systems).

Building BEAMSERV

The BEAMSERV program source files are located in the directory

install_dir/nxnr/bmsrv

on Linux and

```
install_dir\nxnr\bmsrv
```

on Windows (see "Beam Server Source Files").

To build the program, change the working directory to the bmsrv directory and enter the command:

```
nxnr bmsrv build
```

If you do not have write access to *install_dir*/nxn*r*/bmsrv, copy the entire directory to another location, change the working directory to the new location, and issue the command:

```
nxnr ./bmsrv build
```

Using BEAMSERV

NX Nastran is made aware of the beam server by the "gmconn" keyword and an external evaluator connection file. Entries in the connection file for piped communications are formatted as follows:

```
groupname,pipe,pathname
```

where *groupname* is the group name defined on the CONNECT FMS statement and *pathname* is the pathname of the beam server executable.

Note

The group name on the CONNECT FMS statements and in the external evaluator connection file must match exactly, including character case. To use a mixed or lower case group name, the name on the CONNECT FMS statement must be in quote marks; the name in the external evaluator connection file is never quoted.

To use the sample beam server and data file, create the file "samp_eval" with the following line:

```
LOCBMLS,pipe,pathname
```

where *pathname* is the pathname of the beam server built above, e.g., *install_dir*/nxn*r*/*arch*/beamserv or ./beamserv on Linux and *install_dir*\nxn*r**arch*\beamserv or .\beamserv on Windows.

NX Nastran is then run using the following command:

```
nxnr nastran sample gmconn=samp_eval
```

8.3 Building and Using DDLPRT

DDLPRT illustrates the mass retrieval of data from the NX Nastran Access Data Definition Language (DDL) database.

Building DDLPRT

The DDLPRT program source code is in the file "ddlprt.F" (see "NX Nastran Access Source Files"). To build the program, change the working directory to the access directory and type the command:

```
nxnr access ddlprt
```

If you do not have write access to the source directory, *install_dir*/nxn*r*/access on Linux and *install_dir*\nxn*r*\access on Windows, copy the entire directory to another location, change the working directory to the new location, and issue the command:

```
nxnr ./access ddlprt
```

on Linux, or

```
nxnr .\access ddlprt
```

on Windows. Note, the directory specification is required in this circumstance.

Using DDLPRT

DDLPRT is run with the “ddlprt” command. The format of the “ddlprt” command is

```
nxnr ddlprt [ddl_xdb_file] [keywords]
```

If the DDL XDB file is not specified, the program uses the default NX Nastram Access DDL file, *install_dir*/nxn*r*/*arch*/dbc.xdb on Linux and *install_dir*\nxn*r**arch*\dbc.xdb on Windows. The optional keywords are:

print=*print_file*	Default: *ddl_xdb_file*.prt This keyword specifies the name of the print file documenting the format of every NX Nastran Access relation. By default, the print file uses the basename of the input DDL XDB file with the new file type “.prt”. Note, the size of this file is approximately one megabyte.
toc=*table_of_contents_file*	Default: *ddl_xdb_file*.toc This keyword specifies the name of the print file’s table of contents. By default, the toc file uses the basename of the input XDB file with the new file type “.toc”.

To execute the program, enter the command

```
nxnr ddlprt
```

The program displays the filename, version, and compilation date of the DDL file as well as the names of the print and table of contents files. Once these files are generated, the program exits. The print and table of contents files may then be printed once DDLPRT has completed.

8.4 Building and Using DDLQRY

DDLQRY illustrates the interactive retrieval of data from the NX Nastran Access Data Definition Language (DDL) database.

Building DDLQRY

The DDLQRY program source code is in the file “ddlqry.F” (see “NX Nastran Access Source Files”). To build the program, change the working directory to the access directory and type the command:

```
nxnr access ddlqry
```

If you do not have write access to the source directory, *install_dir*/nxn*r*/access on Linux or *install_dir*\nxn*r*\access on Windows, copy the entire directory to another location, change the working directory to the new location, and issue the command:

```
nxnr ./access ddlqry
```

on Linux, or

```
nxnr .\access ddlqry
```

on Windows. Note, the directory specification is required in this circumstance.

Using DDLQRY

DDLQRY is run with the “ddlqry” command. The format of the “ddlqry” command is

```
nxnr ddlqry [ddl_xdb_file]
```

If a file is not specified, the program uses the default NX Nastran Access DDL file, *install_dir*/nxn*r*/*arch*/dbc.xdb on Linux and *install_dir*\nxn*r**arch*\dbc.xdb on Windows.

When running ddlqry, the first prompt will ask you to select a task:

```
Enter Task:(Object,Token,Help,Quit)
```

Selecting ‘Object’, the prompt will show:

```
Enter Object Name (null to quit)
```

After you enter the name of each object, the format of the object is displayed. The program repeats the prompt until a blank line is entered.

8.5 Building and Using DEMO1

DEMO1 prints information about a results database (XDB) file produced by NX Nastran.

Note

The sample program source code is only provided as a simple example illustrating basic concepts. It is not intended to be a complete or usable program.

Building DEMO1

The DEMO1 program source code is in the file “demo1.f” (see “NX Nastran Access Source Files”). To build the program, change the working directory to the access directory and type the command:

```
nxnr access demo1
```

If you do not have write access to the source directory, *install_dir*/nxn*r*/access on Linux or *install_dir*\nxn*r*\access on Windows, copy the entire directory to another location, change the working directory to the new location, and issue the command:

```
nxnr ./access demo1
```

on Linux, or

```
nxnr .\access demo1
```

on Windows. Note, the directory specification is required in this circumstance.

Using DEMO1

DEMO1 is run using the “demo1” command. The installed version of the program is run with the command:

```
nxnr demo1
```

You are prompted for the input database filename.

```
Enter the database path name:
```

Running NX Nastran with a101x.dat (in *install_dir*/nxn*r*/access) produces a101x.xdb that may be used as input to this program.

8.6 Building and Using DEMO2

DEMO2 prints information about a results database (XDB) file produced by NX Nastran.

Note

The sample program source code is only provided as a simple example illustrating basic concepts. It is not intended to be a complete or usable program.

Building DEMO2

The DEMO2 program source code is in the file “demo2.f” (see “NX Nastran Access Source Files”). To build the program, change the working directory to the access directory and type the command:

```
nxnr access demo2
```

If you do not have write access to the source directory, *install_dir*/nxn*r*/access on Linux or *install_dir*\nxn*r*\access on Windows, copy the entire directory to another location, change the working directory to the new location, and issue the command:

```
nxnr ./access demo2
```

on Linux, or

```
nxnr .\access demo2
```

on Windows. Note, the directory specification is required in this circumstance.

Using DEMO2

DEMO2 is run using the “demo2” command. The installed version of the program is run with the command:

```
nxnr demo2
```

You are prompted for the input database filename.

```
Enter the database path name:
```

Running NX Nastran with a101x.dat (in *install_dir*/nxn*r*/access) produces a101x.xdb that may be used as input to this program.

8.7 Building and Using MATTST

MATTST reads a binary format OUTPUT4 matrix.

Note

The sample program source code is only provided as a simple example illustrating basic concepts. It is not intended to be a complete or usable program.

Note

The ILP-64 MATTST executable expects the OUTPUT4 file to be 64–bit integer format, although the default format of OUTPUT4 files written from all exectuble options (LP-64 and ILP-64) is 32–bit integer. If you are using the ILP-64 compiled version of MATTST, you will need to include “param,op4fmt,64” in your original input file as to write the OUTPUT4 file as 64–bit integer, thus making it compatible with the MATTST utility. The alternative is to use the LP-64 executable version of MATTST which expects the OUTPUT4 file to have a 32–bit integer format.

Building MATTST

The MATTST program source code is in the file “mattst.f” (see “Building the Utilities Delivered in Source Form”). To build the program, change the working directory to the util directory and type the command:

```
nxnr util mattst
```

If you do not have write access to the source directory, *install_dir*/nxn*r*/util on Linux or *install_dir*\nxn*r*\util on Windows, copy the entire directory to another location, change the working directory to the new location, and issue the command:

```
nxnr ./util mattst
```

on Linux, or

```
nxnr .\util mattst
```

on Windows. Note, the directory specification is required in this circumstance.

Using MATTST

MATTST is run with the “mattst” command. The installed version of the program is run with the command:

```
nxnr mattst
```

You are prompted for the number of matrices.

```
Please enter the number of matrices:
```

You are prompted for the input filename.

```
Please enter the INPT4 FILENAME:
```

You are prompted for the output binary filename.

```
Please enter the output binary filename:
```

You are prompted for the output text filename.

```
Please enter the output text filename:
```

Running the NX Nastran job “DEMODIR:um54.dat” produces a file, “um54.f11”, that may be used as input to this program.

8.8 Building and Using SMPLR

SMPLR reads a results database (XDB) file produced by NX Nastran.

Note

The sample program source code is only provided as a simple example illustrating basic concepts. It is not intended to be a complete or usable program.

Building SMPLR

The SMPLR program source code is in the file “smplr.f” (see “NX Nastran Access Source Files”). To build the program, change the working directory to the access directory and type the command:

```
nxnr access smplr
```

If you do not have write access to the source directory, *install_dir*/nxn*r*/access on Linux or *install_dir*\nxn*r*\access on Windows, copy the entire directory to another location, change the working directory to the new location, and issue the command:

```
nxnr ./access smplr
```

on Linux, or

```
nxnr .\access smplr
```

on Windows. Note, the directory specification is NX Nastran Access in this circumstance.

Using SMPLR

SMPLR is run using the “smplr” command. The installed version of the program is run with the command:

```
nxnr smplr
```

You are first prompted for the database name.

```
Enter the database name to process:
```

Then you are prompted to enter the access output filename.

```
Enter the access output filename:
```

Running NX Nastran with a101x.dat (see “NX Nastran Access Source Files”) produces a101x.xdb that may be used as input to this program.

8.9 Building and Using TABTST

TABTST reads a binary format OUTPUT2 file (don't confuse this program with RCOUT2, described in “RCOUT2”).

Note

The sample program source code is only provided as a simple example illustrating basic concepts. It is not intended to be a complete or usable program.

Note

The ILP-64 TABTST executable expects the OUTPUT2 file to be in 64-bit integer format, although the default format of OUTPUT2 files written from all exectuble options (LP-64 and ILP-64) is 32-bit integer. If you are using the ILP-64 compiled version of TABTST, you will need to include "param,op2fmt,64" in your original input file as to write the OUTPUT2 file as 64-bit integer, thus making it compatible with the TABTST utility. The alternative is to use the LP-64 executable version of TABTST which expects the OUTPUT2 file to have a 32-bit integer format.

Building TABTST

The TABTST program source code is in the file “tabtst.f” (see “Building the Utilities Delivered in Source Form”). To build the program, change the working directory to the util directory and type the command:

```
nxnr util tabtst
```

If you do not have write access to the source directory, *install_dir*/nxn*r*/util on Linux or *install_dir*\nxn*r*\util on Windows, copy the entire directory to another location, change the working directory to the new location, and issue the command:

```
nxnr ./util tabtst
```

on Linux, or

```
nxnr .\util tabtst
```

on Windows. Note, the directory specification is required in this circumstance.

Using TABTST

TABTST is run with the “tabtst” command. The installed version of the program is run with the command:

```
nxnr tabtst
```

You are prompted for the input filename.

```
Please type the INPUT2 filename:
```

You are prompted for the output filename.

```
Please type the output filename:
```

Running the NX Nastran job “TPLDIR:tabtsta.dat” produces a file, “tabtsta.f11”, that may be used as input to this program.

8.10 Beam Server Source Files

The BEAMSERV program source files are located in the beam server source directory, i.e., *install_dir*/nxn*r*/bmsrv on Linux and *install_dir*\nxn*r*\bmsrv on Windows. This directory is an optional component of the NX Nastran installation. Table 8-1 lists files contained in this directory.

Table 8-1. Beam Server Sample Program Source Files

File	Description
bmsrv	Linux Script to Build the Sample Beam Server Program.
brtucd.F	Source for Sample Beam Server Subroutine BRTUCD.
brtugd.F	Source for Sample Beam Server Subroutine BRTUGD.
brtuid.F	Source for Sample Beam Server Subroutine BRTUID.
brtupd.F	Source for Sample Beam Server Subroutine BRTUPD.
bsbrcd.F	Source for Sample Beam Server Subroutine BSBRCD.
bsbrgd.F	Source for Sample Beam Server Subroutine BSBRGD.
bsbrid.F	Source for Sample Beam Server Subroutine BSBRID.
bsbrpd.F	Source for Sample Beam Server Subroutine BSBRPD.
bsbrt.F	Source for Sample Beam Server Subroutine BSBRT.
bscon.F	Source for Sample Beam Server Subroutine BSCON.
bsgrq.F	Source for Sample Beam Server Subroutine BSGRQ.
bsmsg.F	Source for Sample Beam Server Subroutine BSMSG.
makefile	Makefile to Build the Sample Beam Server Program.
main.c	Source for Sample Beam Server Main Program.
mevbrd.F	Source for Sample Beam Server Subroutine MEVBRD.
msbrcd.F	Source for Sample Beam Server Subroutine MSBRCD.
msbrgd.F	Source for Sample Beam Server Subroutine MSBRGD.
msbrid.F	Source for Sample Beam Server Subroutine MSBRID.
sample.dat	NX Nastran Sample Data File.

8.11 NX Nastran Access Source Files

The NX Nastran Access sample source files are located in the directory *install_dir*/nxn*r*/access on Linux and *install_dir*\nxn*r*\access on Windows. This directory is an optional component of the NX Nastran installation. Table 8-2 lists files contained in this directory.

Table 8-2. NX Nastran Access Sample Program Source Files

File	Description
a101x.dat	NX Nastran Data File.
access	Script to Build NX Nastran Access Sample Programs.
ddlprt.F	Demonstration Database Dictionary Print Program.
ddlqry.F	Demonstration Database Dictionary Query Program.
demo1.F	Source for Sample NX Nastran Database Reader.
demo2.F	Source for Sample NX Nastran Database Reader.
makefile	Makefile to Build NX Nastran Access Sample Programs.
smplr.F	Source for Sample NX Nastran Database Reader.

Appendix A: Glossary

3060	A User Fatal Message indicating that authorization to run NX Nastran has been denied (see "Using the "ugsinfo" Command (Linux)").
6080	A User Warning Message indicating that timing blocks must be generated for your computer (see "Generating a Timing Block for a New Computer").
acct	Accounting file directory, "*install_dir*/acct" on Linux and "*install_dir*/acct" on Windows. Also, the program (*install_dir*/nxn*r*/*arch*/acct on Linux and *install_dir*\nxn*r**arch*\acct.exe on Windows) that updates the current month's accounting data file. See EDSACT for the program source.
architecture RC file	The RC file "*install_dir*/conf/*arch*/nast1rc" on Linux and "*install_dir*\conf*arch*\nast1.rcf" on Windows. See Table 3-1 for a listing of architecture names.
ASSIGN	A File Management Section (FMS) statement that is used to assign physical files to DBsets or FORTRAN files.
authorize	Command line and RC file keyword that is used to set the authorization code required to run NX Nastran.
basename	The part of a pathname exclusive of the directory and file type (e.g., the basename of /temp/myfile.dat. is "myfile").
buffer pool	A disk cache of GINO blocks.
BUFFPOOL	The NASTRAN statement keyword that sets the size of the buffer pool (see "Using the NASTRAN Statement").
BUFFSIZE	One plus the number of words in a GINO physical record. Also, the NASTRAN statement keyword that sets the default buffer size (see "Using the NASTRAN Statement").
conf	The configuration file directory (*install_dir*/conf on Linux and *install_dir*\conf on Windows) contains the system, architecture, and node RC files and other site-specific files.
counted license	A counted license is a FLEXlm license that limits the number of concurrent executions of NX Nastran. Counted licenses always require a FLEXlm license server.

daemon	A Linux program that runs in the background and provides services to the operating system and to users. Daemons are generally started when the system is bootstrapped and terminate when the system shuts down.
dat	Default input data file type.
DBALL	Default DBALL DBSet file type. The DBALL DBSet contains your model and results.
DBSet	Database file set.
DDLPRT	Utility program that prints the contents of the results database (XDB) data definition language database (*install_dir*/nxn*r*/*arch*/dbc.xdb on Linux and *install_dir*\nxn*r**arch*\dbc.xdb on Windows) and illustrates the batch recovery of the data definition language.
DDLQRY	Utility program that prints the contents of the results database (XDB) data definition language database (*install_dir*/nxn*r*/*arch*/dbc.xdb on Linux and *install_dir*\nxn*r**arch*\dbc.xdb on Windows) and illustrates the interactive recovery of the data definition language.
del	Delivery database library,
DEMO	The demonstration problem library (*install_dir*/nxn*r*/nast/demo on Linux and *install_dir*\nxn*r*\nast\demo on Windows) contains a selection of NX Nastran input files that are documented in the NX Nastran Demonstration Problem Manual.
DEMO1	Sample program that prints information from a graphics database file.
DEMO2	Sample program that prints information from a graphics database file.
DMAP	Direct Matrix Abstraction Program, which is the programming language of the NX Nastran solution sequences.
DMP	Distributed Memory Parallel. In NX Nastran, DMP execution is enabled by the “dmparallel” keyword.
doc	Documentation file directory.
EDSACT	Utility program that generates accounting reports. The source for this utility and the accounting file update program are maintained in the same file (*install_dir*/nxn*r*/util/edsact.c on Linux and *install_dir*\nxn*r*\util\edsact.c on Windows).

ESTIMATE	Utility that estimates memory and disk requirement of a data file and make suggestions on improving the performance of NX Nastran.
F04	The F04 file is created by NX Nastran and contains a module execution summary as well as a database information summary. The F04 file has the file type“.f04".
F06	The F06 file is created by NX Nastran and contains the numerical results of the analysis. The F06 file has the file type “.f06".
file locking	A mechanism to prevent multiple jobs from interfering with one another. For example, two jobs attempting to write to the same DBset interfere with one another, whereas two jobs reading the delivery database do not interfere with one another.
file mapping	A mechanism to use the system’s virtual paging system to access a file. NX Nastran can use file mapping to access GINO files. See Table 5-7 for a listing of systems that support file mapping.
FMS	File Management Section of the input file, which is used to attach and initialize DBsets and FORTRAN files.
gentim2	NX Nastran job that determines the timing constants for your computer.
GINO	The NX Nastran database subsystem.
GINO block	A block of data transferred by GINO.
HEATCONV	Utility program that converts pre-MSC.Nastran V68 heat-transfer data files to the MSC.Nastran Version 68 format.
HIPPI	High Performance Parallel Interface. An ANSI standard (ANSI X3T9.3 document number X3T9.3/90-043, 1990) interface used in high-performance environments.
IEEE	Institute of Electrical and Electronics Engineers, Inc. A professional society. The floating point formats and, to a lesser extent, algorithms used on most NX Nastran computers are defined by IEEE Standard 754.
INCLUDE	A general NX Nastran input file statement that inserts an external file into the input file. INCLUDE statements may be nested.
INIT	The INIT statement is part of the File Management Section (FMS) and is used to create a temporary or permanent DBset.
large file	A file on a 32-bit system that can be 2 gigabytes or larger. All files on a 64-bit system can be large files. See Table 5-7 for a listing of systems that support large files.

local RC file	The RC file ".nast1rc" on Linux and "nast1.rcf" on Windows in the directory containing the input data file.
LOG	The LOG file is created by NX Nastran and contains system information as well as system error messages. The LOG file has the file type ".log".
MASTER	Default MASTER DBSet file type. The MASTER DBSet contains the names of other database members and indices.
MATTST	Sample program that reads the OUTPUT4 matrix files.
memory	Command line keyword that is used to define the amount of memory allocated for open core.
MPI	Message Passing Library. An industry-standard library for message passing programs.
MPL	The module properties list is a table that defines the properties of DMAP modules.
MSGCMP	Utility program that compiles a text file to create a message catalog.
NAO	The Network Authorization Option of NX Nastran.
ndb	Default neutral-format results database file type.
neu	Default neutral-format plot file type. Only created by NEUTRL.
NEUTRL	Utility program that converts binary plot (.plt) files to neutral plot (.neu) files.
node RC file	The RC file "*install_dir*/conf/net/nodename/nast1rc" on Linux and "*install_dir*\conf\net\nodename\nast1.rcf" on Windows.
NUSR	The node-locked license enforcement of the maximum number of users concurrently running NX Nastran. See "Enabling Account ID Validation" for additional information.
NX Nastran ACCESS	FORTRAN-callable subroutine library that reads and writes results database (XDB) files.
on2	Default neutral-format OUTPUT2 file type.
op2	Default binary-format OUTPUT2 file type.
open core	Amount of working memory in words.
OPTCONV	Utility program that converts pre-MSC.Nastran V68 optimization and design-sensitivity data files to the MSC.Nastran Version 68 format.

pch	Default punch file type.
PLOTPS	Utility program that converts binary (.plt) or neutral (.neu) plot files to PostScript (.ps) files.
plt	Default binary-format plot file type.
ps	Default PostScript plot file type.
RC file	Runtime configuration file that is used by NX Nastran to control execution parameters.
RCOUT2	Utility program that converts a neutral OUTPUT2 (.np2) file to a binary OUTPUT2 (.op2) file.
RECEIVE	Utility program that converts neutral results database (.neu) files to binary results database (XDB) files.
SCR300	Default SCR300 DBSet file type.
SCRATCH	Default SCRATCH DBSet file type.
sdir	Keyword that is used to set the directory for temporary scratch files produced by NX Nastran.
SMEM	Scratch memory area for memory-resident database files.
smemory	Command line keyword to set SMEM.
SMP	Shared Memory Parallel. In NX Nastran, SMP execution is enabled by the “parallel” keyword.
SMPLR	Sample program that reads graphics database files.
SSS	Structured Solution Sequences. The delivery database files (SSS.MASTERA, SSS.MSCSOU, and SSS.MSCOBJ) are found in “*install_dir*/nxn*r*/*arch*” on Linux and “*install_dir*\nxn*r**arch*” on Windows; the source files are found in “*install_dir*/nxn*r*/nast/del” on Linux and “*install_dir*\nxn*r*\nast/del” on Windows.
SSSALTER	Additional alter and error corrections library, “*install_dir*/nxn*r*/misc/sssalter” on Linux and “*install_dir*\nxn*r*\misc\sssalter” on Windows.
SYS	An ASSIGN statement parameter that is used to specify special machine-dependent information. File locking and file mapping of database files are controlled through the SYS parameter.
sysfield	The global SYS parameter that can be specified on the command line or in an RC file.
system RC file	The RC file “*install_dir*/conf/nast1rc” on Linux and “*install_dir*\conf\nast1.rcf” on Windows.

SYSTEM(x)	System cells that are used by NX Nastran to control analysis parameters.
TABTST	Sample program that reads binary-format OUTPUT2 files.
TPL	The test problem library (TPL, *install_dir*/nxn*r*/nast/tpl on Linux and *install_dir*\nxn*r*\nast\tpl on Windows) contains a general selection of NX Nastran input files showing examples of most of the NX Nastran capabilities. In general, these files are not documented.
TRANS	Utility program that converts binary results database (XDB) files to neutral results database (.neu) files.
type	The part of the pathname exclusive of the directory and basename (e.g., the file type of myfile.dat is ".dat").
UFM	A User Fatal Message that describes an error severe enough to terminate the program.
UFM 3060	A User Fatal Message indicating that authorization to run NX Nastran has been denied (see "Using the "ugsinfo" Command (Linux)").
UIM	A User Information Message that provides general information.
user RC file	The RC file "$HOME/.nast1rc" on Linux and "%HOMEDRIVE%%HOMEPATH%/nast1.rcf" on Windows.
util	Utility program library, "*install_dir*/nxn*r*/util" on Linux and "*install_dir*\/nxn*r*\util" on Windows.
UWM	A User Warning Message that warns of atypical situations. You must determine whether a problem exists in the analysis.
version	A file is "versioned" by appending a dot followed by a version number to the file's name. The latest version of a file does not have a version number, all earlier versions do, with the oldest having the smallest version number and the latest having the highest version number.
XDB	The XDB file is created by NX Nastran and contains results information for use by various post-processing programs. See the "POST" parameter in "Parameters" in the *NX Nastran Quick Reference Guide* for further information on generating XDB files. XDB files are not versioned. The XDB file has the file type ".xdb".

Appendix B: Keywords and Environment Variables

The following is a complete list of the keywords that may be used on the command line or placed into RC files as appropriate.

Keywords that use yes/no values accept partial specification and case-independent values. For example, “yes” may be specified as “y”, “ye”, or “yes” using uppercase or lowercase letters.

acct acct=yes,no Default: No

Indicates solution accounting is to be performed. The new “lock” keyword may be used to ensure that all jobs have solution accounting enabled.

For example, the following RC file lines force all jobs to use accounting:

Example:

```
acct=yes
lock=yes
```

The first line turns accounting on. The second line ensures accounting is on for every job; see the “lock” keyword for more details.

acdata acdata=*string* Default: None

Specifies site defined accounting data. See your system administrator to determine if and how this keyword is to be used. See “Enabling Account ID and Accounting Data” for additional information.

acid acid=*string* Default: None

Specifies the site defined account ID for this job. See your system administrator to determine if and how this keyword is to be used. See “Enabling Account ID and Accounting Data” for additional information.

acvalid acvalid=*string* Default: None

This keyword can only be set in the command initialization file, see the sections titled “Enabling Account ID Validation” and “Customizing the Command Initialization File”.

Indicates account ID validation is to be performed. If “acvalid” is not defined, or is null, then no checks are made of the account ID. If “acvalid” is defined, then account ID validation is performed. “Enabling Account ID and Accounting Data” contains more information on defining this keyword.

after (Linux) after=*time* Default: None

Holds the job’s execution until the time specified by time. See the description of the “at” command in your system documentation for the format of time.

Example: `nxnr nastran example after=10:00`

The job is held until 10:00 AM.

append append=yes,no Default: No

Combines the F04, F06, and LOG files into a single file after the run completes. If “no” is specified, the files are not combined. If “yes” is specified, the files are combined into one file with the type “.out”.

Example: `nxnr nastran example append=yes`

The F04, F06, and LOG files are combined into a file named “example.out”.

application

application=NASTRAN

Specifies the application to be run.

This keyword should always be set to “NASTRAN”, and may only be specified on the command line or in the command initialization file. See “Customizing the Command Initialization File”.

authinfo

authinfo=*number* Default: 0

Specifies the amount of information written to the LOG during authorization processing. Values greater than zero indicate additional information is to be written.

authorize

authorize=*spec* Default: Linux: port@host Windows: port@host

Selects the licensing method for NX Nastran. The spec can take on several forms. They include:

authorize=FLEXlm-license-spec FLEXlm licensing has been selected. See “Automatically Starting a FLEXlm Server on Linux Platforms” for information on specifying a FLEXlm license.

authorize=*pathname* This specifies a FLEXlm license file. If only a directory is specified, the program assumes that either “authorize.dat” or “license.dat” is in the specified directory.

Example: `nxnr nastran example auth=myauthfile`

The job runs using the node-locked authorization code in “myauthfile”.

authqueue

authqueue=*number of minutes* Default: 0

When NX Nastran attempts to run, but cannot because a license is unavailable, the job will be retried every minute up to the value of the AUTHQUEUE keyword.

batch

(Linux)

batch=yes,no Default: Yes

Indicates how the job is to be run. If “yes” is specified, the job is run as a background process. If “no” is specified, the job is run in the foreground. If the “aft” or “queue” keywords are specified, the batch keyword is ignored. Jobs submitted with “batch=yes” will run under nice(1).

Note: If the job is already running in an NQS or NQE batch job, the default is “no”.

Example: `nxnr nastran example batch=no`

The job is run in the foreground.

bpool

bpool=*value* Default: 37

Specifies the number of GINO and/or executive blocks that are placed in buffer pool.

Example: `nxnr nastran example bpool=100`

Space for 100 GINO buffers is reserved for the buffer pool.

buffsize

buffsize=*value* Default: 8193

Specifies the physical record size, in words (1 word = 4 bytes using the LP-64 executables, 1 word = 8 bytes using ILP-64 executables), of all NX Nastran DBsets except those specified with INIT statements and MSCOBJ. The physical I/O size is BUFFSIZE-1 words.

If “buffsize=estimate” is specified, ESTIMATE will be used to determine value.

See “Estimating BUFFSIZE” for recommended BUFFSIZE values based on model size.

BUFFSIZE must reflect the maximum BUFFSIZE of all DBsets attached to the job including the delivery database, which is generated with a BUFFSIZE of 8193. If you generate your own delivery database, this default may be different. The maximum value of BUFFSIZE is 65537 words. BUFFSIZE must be one plus a multiple of the disk block size. The disk default block size may be determined with the “system” special function described in “Using the Help Facility and Other Special Functions”; specific block size information may be obtained from your system administrator.

Example: `nxnr nastran example buffsize=16385`

The BUFFSIZE is set to 16385 words.

config

config=*number* Default: Computer dependent

Specifies the configuration (CONFIG) number used by NX Nastran to select timing constants. You can change this value to select the timing constants of a different computer model. A configuration number of zero is considered undefined by the nastran command. See “Defining a Computer Model Name and CONFIG Number” and “Generating a Timing Block for a New Computer” for additional information.

cpulimit (Linux)

cpulimit=*CPU time* Default: None

Note: The following capability is dependent upon the queue submission commands defined by the “submit” keyword and your queuing system. The capability or examples may not work on your system.

Specifies the maximum amount of CPU time that the complete job is permitted to use when the “queue” keyword is used. This time includes the execution of the driver program, the NX Nastran executable, plus any commands specified by the “pre” and “post” keywords. See your system’s queuing documentation for the format of cpulimit.

The value can be specified as either “hours:minutes:seconds”, “minutes:seconds”, or “seconds”; it will always be converted to seconds by the nastran command.

Example: `nxnr nastran example \ queue=small cpulimit=60`

This example defines the maximum CPU time for the complete job as 60 seconds.

Example:

```
nxnr nastran example \
queue=small cpu=1:15:0
nxnr nastran example queue=small cpu=75:0
nxnr nastran example queue=small cpu=4500
```

These examples all define the maximum CPU time for the complete job as one hour and fifteen minutes.

cpyinput cpyinput=0,1 Default: 0

Indicates the input data file is to be copied to an temporary file before processing. Setting cpyinput=1 will emulate the behavior of copying the file, this will consume additional time and disk resources.

dbs dbs=*pathname* Default: .

Creates database files (see Table 5-6) using an alternate file prefix. If “dbs” is not specified, database files are created in the current directory using the basename of the input data file as the prefix. If the “dbs” value is a directory, database files are created in the specified directory using the basename of the input data file as the filename.

Note: If “dbs” is specified and “scratch=yes” is specified, a warning will be issued and “scratch=no” assumed.

In the following examples, assume the current directory includes sub-directories “mydir” and “other”, and that an “example.dat” exists in both the current directory and “other”. That is, ./example.dat, ./mydir, ./other, and ./other/example.dat exist on Linux; and .\example.dat, .\mydir, .\other, and .\other\example.dat exist on Windows.

Example: `nxnr nastran example`

Database files are created in the current directory with the name “example”, e.g., ./example.DBALL on Linux; and .\example.DBALL on Windows.

Example: `nxnr nastran other/example`

Database files are created in the “other” directory with the name “example”, e.g., ../other/example.DBALL on Linux and .\other\example.DBALL on Windows.

Example: `nxnr nastran example dbs=myfile`

Database files are created in the current directory with the name “myfile”, e.g., ./myfile.DBALL on Linux and .\myfile.DBALL on Windows.

Example: `nxnr nastran example dbs=mydir`

Database files are created in the mydir directory with the name “example”, e.g., ./mydir/example.DBALL on Linux and .\mydir\example.DBALL on Windows.

Example: `nxnr nastran example dbs=mydir/myfile`

Database files are created in the mydir directory with the name “myfile”, e.g., ./mydir/myfile.DBALL on Linux and .\mydir\myfile.DBALL on Windows.

Example: `nxnr nastran example dmp=4 host=a:b:c:d dbs=/aa:/bb:/cc:/dd`

This example will set the “dbs” directory to “/aa” on host a, “/bb” on host b, “/cc” on host c, and finally “/dd” on host d.

Note: Using distinct per-task database directories can significantly impact the elapsed time performance of DMP jobs on SMP and NUMA systems.

delete delete=yes,no,all,jid Default: No

Note: This keyword is only intended to be used when NX Nastran is running in server mode or is embedded within another application. The deletion occurs before the post commands are run.

Unconditionally delete files after an NX Nastran job completes. Specifying “delete=yes” will delete the F04, F06, and LOG file when the job completes; “delete=all” will delete the F04, F06, LOG, NDB, OP2, PCH, PLT, and XDB files when the job completes. You can also specify a list of file types, e.g., “delete=f04,log,plt” will only delete the F04, LOG, and PLT files.

Example: `nxnr nastran example delete=op2,plt`

After the NX Nastran job has completed, the “example.op2” and “example.plt” files will be unconditionally deleted. These files are normally kept if they are not empty.

delivery delivery=[*directory path*]filename Default: NXNDEF

Specifies an alternate delivery database option. See “Creating and Attaching Alternate Delivery Databases” for further information on alternate delivery databases.

The *directory path* is only needed when the file location is not the run directory (location of input file). When used, it can be entered relative to the run directory, or as an absolute path.

The default “NXNDEF” indicates the standard NX Nastran delivery database.

Example: `nxnr nastran example del=mysss`

The job runs using a solution sequence from the user supplied delivery database “mysss.MASTERA”.

diag diag=flag,flag,... Default: None

Sets NX Nastran diagnostics. This keyword may also be set with the DIAG Executive Control Statement. See “DIAG” in the *NX Nastran Quick Reference Guide* for information on the default value and legal values for this keyword. The diagnostics set using this keyword are in addition to any diagnostics set with the DIAG statement in the input file.

Example: `nxnr nastran example diag=5`

The NX Nastran job is run with DIAG 5 set.

dmparallel dmparallel=number Default: 0

Specifies the number of tasks for a Distributed Memory Parallel (DMP) analysis. This value may only be set on the command line.

The value must be null or zero to cancel DMP processing, or a number greater than zero to enable DMP processing.

For a Hierarchic Domain Parallel Normal Modes via Lanczos (HDMP) analysis, the keyword nclust must be used in conjunction with dmparallel. See keyword “nclust” .

Example: `nxnr nastran example dmp=4`

The job is run with four DMP tasks.

dskco dskco=*value* Default: 1

Allows you to define a factor to scale total disk estimates. This scale factor is applied before the "dskmin" value, that provides a lower bound for total disk estimates.

Example: `nxnr estimate example dskco=2`

This doubles the total disk estimate and then applies the "dskmin" lower bound.

Example: `nxnr estimate example dskco=0.5`

This will halve the total disk estimate. An estimate less than the lower bound specified by "dskmin" will be set to the lower bound.

dskmin dskmin=*value* Default: 1mb

Allows you to define the lower bound for all total disk estimates. This bound is applied after the "dskco" value, that multiplies the actual estimate by a "conservatism" factor.

Example: `nxnr estimate example dskmin=2mb`

This will set the minimum total disk estimate to 2 MB.

endian endian=*value*

The endian format of binary results files becomes important in cases when you run NX Nastran on one type of workstation, then transfer your binary files to another for post processing. By default, the endian format of OUTPUT2 and OUTPUT4 files depends on the workstation type which created them. In general, Intel and Linux workstations use the Little Endian format, and Linux workstations use Big Endian. The endian keyword inputs are:

“endian = big” writes output2 and output4 files as big endian binary format.

“endian = little” writes output2 and output4 files as little endian binary format.

executable executable=*pathname* Default: Computer dependent

Specifies the name of an alternate solver executable. This keyword overrides all architecture and processor selection logic. If a directory is not specified by pathname and the file does not exist in the current directory, the default architecture directory is assumed.

Example: `nxnr nastran example exe=analysis.um`

The job runs using the executable “analysis.um”. Since a directory was not specified, this file must exist in either the current directory or *install_dir*/nxn*r*/*arch* on Linux or *install_dir*\nxn*r**arch* on Windows.

f04 f04=*number* Default: 4

Specifies FORTRAN unit number for standard output file. See the “nastran Command and NASTRAN Statement” section of the *NX Nastran Quick Reference Guide* for more information.

f06 f06=*number* Default: 6

Specifies FORTRAN unit number for standard output file. See the “nastran Command and NASTRAN Statement” section of the *NX Nastran Quick Reference Guide* for more information.

fbsmem fbsmem=*number* Default: See the *NX Nastran Quick Reference Guide*

Reserves memory for faster solution of the Lanczos method of eigenvalue extraction. This keyword may also be set with the “sys146” command line keyword. See the *NX Nastran Quick Reference Guide* for information on the default value and legal values for this keyword.

fbsopt fbsopt=*number* Default: See the *NX Nastran Quick Reference Guide*

Selects the forward-backward substitution methods. This keyword may also be set with the “sys70” command line keyword. See the *NX Nastran Quick Reference Guide* for information on the default value and legal values for this keyword.

gmconn gmconn=*pathname* Default: None

Specifies the name of the external evaluator connection file. External geometric and bar or beam element evaluators may be specified. Also, see “Using BEAMSERV” for information on running an NX Nastran job using a beam server.

Example: `nxnr nastran example gmconn=mybeamserver`

The job is run with the external evaluators specified in “mybeamserver”.

gpart gpart=*number* Default: 1

Selects the geometry partitioning method for Hierarchic Domain Parallel.

= 1 (default) Matrix graph partitioning technique is applied. This technique can be used with models containing virtual mass (MFLUID), or when coupled matrices (acoustics) are applied. It should also perform better on models with extremely large numbers of MPCs (spot weld models). It is available for HDMP normal modes (SOL 103) and HDMP modal response solutions (SOLs 111, 112).

= 0 Finite element model partitioning technique applied. Available for DMP statics (SOL 101) and HDMP normal modes solutions (SOL 103).

hostovercommit hostovercommit=yes,no Default: No

Allows this job to assign more tasks to a host than processors. This does not prevent other NX Nastran jobs or users from using the processors. See also the “hosts” keyword below.

If “hostovercommit=no” is specified, at most one task will be assigned for each processor on the host, i.e., a four processor system can only have four tasks assigned.

If “hostovercommit=yes” is specified, tasks are assigned to hosts in a round-robin order until all tasks are assigned, without regard to the number of processors on the host.

Note: Assigning more tasks to a host than it has processors will impact the elapsed-time performance of your DMP job.

In the following examples, assume that host1 and host2 each have two processors.

Example:

```
nxnr nastran example dmp=6 \
 hosts=host1:host2 hostovercommit=no
```

The job will not be started because a total of only four processors are available on host1 and host2.

Example:

```
nxnr nastran example dmp=6 \
 hosts=host1:host2 hostovercommit=yes
```

The job will be allowed to start, with three tasks each assigned to host1 and host2.

hosts=*host*:*host*:...

hosts

hosts=*host*;*host*;... Default: See text

hosts=*filename*

Defines the list of candidate hosts to be used for a DMP analysis. This list is scanned in a round-robin order until all tasks have been assigned to a host. If "hostovercommit=no" is specified, at most one task will be assigned for each processor on the host, i.e., a four processor system can only have four tasks assigned.

Multiple hosts are specified in the standard manner for the PATH environment variable, that is "hosts=*host1*:*host2*:..." on Linux and "hosts=*host1*;*host2*;..." on Windows.

See "Overview of Distributed Memory Parallel (DMP) Jobs" for additional information.

In the following examples, assume that the current host, host1, and host2 each have two processors.

Example: `nxnr nastran example dmp=2`

Example:

```
nxnr nastran example dmp=3 \
 hosts=host1:host2
```

The first and third tasks will be assigned to host1, the second task will be assigned to host2.

Example:

```
nxnr nastran example dmp=3 \
 hosts=myhostfile
```

The file ./myhostfile on Linux and .\myhostfile on Windows will be read to determine the list of hosts to use.

ishellext

ishellext=*value*,*value*,... Default: See text.

Defines command processor associations for ISHELL executables. Each value is specified as "file-type=processor" where processor is the executable used by NX Nastran to execute an ISHELL program with the specified file-type. See "Running an ISHELL Program" for information on using an ISHELL program and the default list of processors.

Specify two consecutive quotes, e.g., ishellext=ksh=" to specify a null processor, that is, to directly execute the ISHELL program. Note, you will need protect the quotes from the shell if specified on the command line.

Specify a null file-type to define a processor for files without a file type.

Specify ".=\"\" to specify a null *file-type* and a null *processor*.

Specifying a *file-type* already defined in the table will replace the previous entry; specifying a *file-type* not yet defined in the table will append the new entry to the end of the table, that is, it will be processed last.

Note: On Windows, all executable files must have a non-null file-type. This is why “TPLDIR:QAISHELL” executable cannot be used on Windows, but “TPLDIR:qaishell.pl” can.

1. On Windows, it may be necessary to define “CMD.EXE” as the processor for certain “.EXE” files, e.g., 16-bit compiled Basic program. This can be done with “ishellext=exe=cmd”

2. Up to twenty associations can be defined.

This keyword may also be set with the *NXN_ISHELLEXT* environment variable. The environment variable overrides the RC files; the command line overrides the environment variable.

Example:

```
nxnr nastran example \
 ishellext=tcl=wish,sh=ksh
```

This example will add one association and replace another. If the ISHELL program name exists with the file type “.tcl”, the wish executable will be used; if the ISHELL program name exists with the file type “.sh”, the ksh executable will be used. Since neither processor specification included a pathname component, the system PATH will be searched for the executables.

ishellpath

ishellpath=*value*:*value*:..
ishellpath=*value*;*value*;..

Default: See text.

Defines a list of directories to search for the ISHELL program if a suitable ISHELL program doesn’t exist in the current working directory. If this list is exhausted before finding a suitable ISHELL program, the standard PATH is searched. Multiple paths are specified in the standard manner, that is “ishellpath=/dir1:/dir2:...” on Linux and “ishellpath=\dir1;\dir2;...” on Windows.

If you have not set a value for “ishellpath”, the value will be set to the directory containing the input data file, this automatically handles the common case where the ISHELL program is located in the same directory as the input data file referencing it.

This keyword may also be set with the NXN_ISHELLPATH environment variable. The environment variable overrides the RC files; the command line overrides the environment variable.

Example: `nxnr nastran TPLDIR:qaishell`

Assuming no RC file set “ishellpath” and the environment variable NXN_ISHELLPATH was not defined, the “ishellpath” value will be set to the directory referenced by “TPLDIR:”. NX Nastran will attempt to locate the ISHELL program in the current working directory, the TPL directory, or in the PATH.

Example: `nxnr nastran example ishellpath=bin`

This example assumes either the current working directory or the bin subdirectory contains the ISHELL program

jid jid=*pathname* Default: None

Specify the name of the input data file. An input file must be defined on the command line. Any command line argument that does not have a keyword is assumed to be the input file; only the last filename is used.

Example: `nxnr nastran this that example`

The input file “example.dat” is used; the tokens “this” and “that” are ignored.

Note: If the input file is specified as “example” and the files “example.dat” and “example” both exist, the file “example.dat” will be chosen. In fact, it is *impossible* to use a file named “example” as the input data file if a file named “example.dat” exists.

jidpath jidpath=*path-spec* Default: None

Specify a list of directories to search if the input data file or any INCLUDE file does not specify a pathname component and does not exist in the current directory.

This keyword may also be set by the NXN_JIDPATH environment variable. The environment variable overrides the RC files, and the command line overrides the environment variable.

Linux example: `nxnr nastran example jidpath=$HOME`

Windows Example:

```
nxnr nastran example \
jidpath=%HOMEDRIVE%%HOMEPATH%
```

These find the file “example.dat” or “example” if it is located in either the current working directory or your home directory.

Multiple directories are specified using the standard syntax for the PATH environment variable. For example:

Linux example:

```
nxnr nastran example \
 jidpath=/models/a:/models/b
```

Windows Example:

```
nxnr nastran example \
 jidpath=\models\a;\models\b
```

Your specification of this value in RC files can include environment variable references. On Linux, use the standard shell “$name” or “${name}” syntax; on Windows use the standard “%name%” syntax.

jidtype jidtype=*file-type* Default: dat

Specify an alternate default file-type of the input data file and any INCLUDE files.

Example: `nxnr nastran example jidtype=bdf`

This example will set the default file type to “bdf”, i.e., the nastran command will look first for a file named “example.bdf”, and if that is not found for the file “example”; if neither file is found, an error will be reported.

If you have not defined a value for the “jidtype” keyword, the nastran command will set the keyword to the actual file type of the input data file.

Example: `nxnr nastran example.bdf`

The nastran command looks for “example.bdf.dat”, if that file does not exist, it then looks for “example.bdf”. Assuming that file exists, and no other value for “jidtype” has been defined, the nastran command sets “jidtype=bdf”.

lock lock=*keyword* Default: None

The “lock” keyword can be used by a site or a user to prevent modification of a keyword’s value.

For example, the following RC file lines will force all jobs to use accounting by setting the “acct” keyword on and then preventing the keyword from being changed later in an RC file, or on the command line:

Example:

```
acct=yes
lock=acct
```

Once these lines are read, any attempt to set the “acct” keyword later in the same RC file, in an RC file read after this file, or on the command line will be silently ignored. See “Setting RC File Keywords” for information on RC file and command line processing.

The “lock” keyword may appear anywhere a keyword is accepted. The lock keyword itself can be locked with “lock=lock”.

Example:

```
authorize=license-spec
lock=authorize
```

Once these lines are read, any attempt to set the “authorize” keyword later in the same RC file, in an RC file read after this file, in the environment via “NXN_LICENSE_FILE,” or on the command line will be silently ignored.

lsymbol lsymbol=*name*=value Default: None

Assigns a value to a local symbolic name. The syntax and usage is identical to the symbol keyword syntax except that symbolic names defined using the lsymbol keyword will not be passed to remote hosts, i.e., to hosts specified with either the “hosts” or "node" keywords.

When neither “hosts” or "node" are specified, symbolic names defined with the symbol keyword are treated as local. In this case, symbolic names defined using the lsymbol keyword will over-ride symbolic names specified using the symbol keyword.

massbuf massbuf=*number* Default: See the *NX Nastran Quick Reference Guide*.

Sets half the number of buffers to set aside for storing the mass matrix in memory. This keyword may also be set with the “sys199" command line keyword. See the *NX Nastran Quick Reference Guide* for information on the default value and legal values for this keyword.

maxnode maxnode=*number* Default: Value of dmparallel parameter

memmin memmin=*value* Default: 16mb

Allows you to define the lower bound for all memory estimates. This bound is applied after the "memco" value, that multiplies the actual estimate by a "conservatism" factor.

Example: `nxnr estimate example memmin=8mb`

This will set the minimum memory estimate to 8 MB.

This keyword may also be set with the MP_NODES environment variable. The environment variable overrides the RC files; the command line overrides the environmental variable.

memory memory=*size* Default: estimate

See Determining Resource Requirements for information on estimating a job's memory requirements.

Specifies the amount of open core memory to allocate. If "memory=estimate" is specified, ESTIMATE will be used to determine size. Otherwise, the *size* is specified as a memory size, see "Specifying Memory Sizes".

If you do not assign a value to the "memory" keyword, and ESTIMATE runs but fails to provide an estimate, the value specified by the "memorydefault" keyword will be used. If "memorydefault" is null, the nastran command will issue a fatal error and the job will end.

Example: `nxnr nastran example memory=25mw`

The job is run using an open core memory size of 25 MW, or 25600 KW, or 26214400 words.

Example: `nxnr nastran example memory=0.5xPhysical`

If run on Windows, the job is run using an open core memory size of half the computer's physical memory. If run on Linux and the computer's physical memory was not defined using the "s.pmem" keyword, the job will fail.

memorydefault memorydefault=*size* Default: 10mw

Specifies the default memory size if a null value was defined for the "memory" keyword, or "memory=estimate" was defined and the ESTIMATE utility failed to provide an estimate.

Note: If a null value is defined for "memorydefault" and it is used as described above, the job will not start.

memorymax memorymax=*size* Default: Linux: 0.8*physical; Windows: 1.2*physical

Specifies the maximum memory size that may be requested. Any request in excess of this will be limited to the "memorymaximum" value. See "Specifying Memory Sizes" for NX Nastran's maximum memory limits.

Note: If size includes a reference to "physical" or "virtual", and the value is not known, the "memorymaximum" value will be silently ignored.

In the following examples, assume "memorymaximum=1gb" was set in an RC file.

Example: `nxnr nastran example memory=900mb`

The job is run using an open core memory size of 900MB.

Example: `nxnr nastran example memory=1200mb`

The job is run using an open core memory size of 1GB, i.e., the "memorymaximum" value set in the RC file.

mergeresults mergeresults=yes,no Default: Yes

Specifies the results from each DMP task are to be merged into the standard files from the master host.

Setting “mergeresults=yes” will cause the output from all tasks to appear in the output files for the master task. That is, as if the analysis were run with one task.

Setting “mergeresults=no” will cause the output from each tasks to appear task-specific output files. That is, each file will need to be examined to get all results.

Note: If “mergeresults=no” is specified in a static run the results of the individual domains will not be sent back to the master and the system solution will not be obtained. The keyword “mergeresults” has no affect on a solution 103 or 111 run. The only circumstances where “mergeresults=no” is recommended is where xdb files are requested and intended to be attached using MSC.Patran in solution 108. In solution 108, if “mergeresults=no” is specified and “slaveout=yes” is not specified, then the results of the slave processors will be lost. In solution 108, it is possible to get a through-put advantage by saving communication between the master and slaves when “mergeresults=no” and “slaveout=yes” is specified.

mio mio=yes,no Default: no

Optionally selects an asynchronous library called “MIO (modular I/O)” for x86_64 Linux platform. The MIO library addresses the tuning needs for I/O by using a caching mechanism. This caching mechanism bypasses the operating system I/O caching or buffering.

x86_64 Linux

When mio=yes, you can set either (a) “MIO_TOOLS_DIR” environment variable to point to the location of the mio tools directory, or (b) set the environment variable MIO_LIBRARY_PATH to point to the appropriate MIO library directory.

When mio=no (default), NX Nastran sets the environment variable, MIO_FILES to the following value:

```
"*SCR*:*scr* [trace/stats/mbytes | pf/cache=2g/page=2m/pref=2 | trace/stats/mbytes
```

mio_cachesize mio-cachesize=*size* Default: 0

mpyad mpyad=*number* Default: See the *NX Nastran Quick Reference Guide*.

Selects/deselects multiplication method selection. This keyword may also be set with the “sys66" command line keyword. See the *NX Nastran Quick Reference Guide* for information on the default value and legal values for this keyword.

msgcat msgcat=*pathname* Default:

Linux: *install_dir*/nxn*r*/*arch*/analysis.msg

Windows: *install_dir*\nxn*r**arch*\analysis.msg

The “msgcat” keyword specifies an alternate message catalog containing the message text used for many NX Nastran messages. A site or user can modify the message file to include message text that is more appropriate to their operations, compile the new catalog using the MSGCMP utility, and invoke the new catalog using this keyword.

Example: `nxnr nastran example msgcat=mycat.msg`

This example will use the file “mycat.msg” as the message catalog. See the sections titled “Customizing the Message Catalog” and “MSGCMP” for additional information.

Note: Message catalogs are computer-dependent, “Binary File Compatibility”, identifies the systems that are binary compatible; binary compatible systems can use the same message file.

nastran nastran keyword=*value* Default: None

Specifies a value for the NASTRAN statement.

Note: This keyword can only be specified in an RC file. If the last character of the keyword value is a comma, or a quote or parenthetic expression is open, the next line in the RC file is considered a continuation. The statement will continue until the quote or parenthetic expression is closed and a line that is not ended by a comma is found.

nclust nclust=*number* Default: 0

Specifies the number of frequency segments when using the hierarchic dmp normal modes analysis (HDMP) capability. In order to improve the balance of the frequency domain decomposition, the existing ALPHA tuning value of the EIGRL continuation card may be used.

The number of processors used in a HDMP solution is specified with the existing dmparallel keyword (which can be abbreviated to “dmp”). The number of geometry partitions is the value of the dmparallel keyword divided by the value of the nclust keyword. The result of the division of the dmparallel keyword and the nclust keyword must be an integer greater than or equal to 2, and a power of 2. For example, 2, 4, 8, 16, etc. are valid results.

ncmd ncmd=*command* Default: print msg | write user tty

Specifies an alternate job completion notification command (see the “notify” keyword). If this keyword is being set on the command line, and command contains embedded spaces, enclose command in quotes.

If the specified command contains the two-character sequence {}, the sequence is replaced by the text “NX Nastran job name completed”.

Note: The following example may not work on your system.

Example:

```
nxnr nastran example notify=yes \
 ncmd="print {} | \
 mail -s {} $(whoami)"
```

At the end of the job, mail is sent to the user submitting the job. The braces in the “ncmd” value are replaced by the job completion text, and the modified command is run:

```
print "NX NASTRAN job example completed" |\
mail -s "NX NASTRAN job example completed" user
```

Windows example: `nxnr nstran example “ncmd=echo done”`

The word “done” will be printed in the command window when the job completes.

newhess newhess=*number* Default: See the *NX Nastran Quick Reference Guide*.

Requests the complex eigenvalue method. This keyword may also be set with the “sys108" command line keyword. See “EIGC” in the *NX Nastran Quick Reference Guide*, and the *NX Nastran Numerical Methods User’s Guide* for information on the default value and legal values for this keyword.

news news=yes,no,auto Default: Yes

Displays the news file (*install_dir*/nxn*r*/nast/news.txt on Linux and *install_dir*\nxn*r*\nast\news.txt on Windows) in the F06 file. If “auto” is specified, the news file is only displayed if it has been modified since the last time it was displayed for you. If “yes” is specified, the news file is displayed in the F06 file regardless of when it was last changed. If “no” is specified, the news file is not displayed in the F06 file.

Example: `nxnr nastran example news=yes`

The news file is displayed in the F06 file after the title page block.

Note: The news file can also be displayed on the terminal by using the command:

```
nxnr nastran news
```

node node=*nodename* Default: None

Executes the job on the specified Linux node.

See “Running a Job on a Remote System (Linux)” for additional information.

Use the “username” keyword to specify an alternate user name on the remote node. This keyword may only be specified on the command line.

Example: `nxnr nastran example node=othernode`

The job is run on the computer named “othernode”.

Note: This capability doesn't permit Windows systems to run jobs on other Windows computers.

notify notify=yes,no Default: Yes

Sends notification when the job is completed. See the “ncmd” keyword to define an alternate notification command.

Note: If the job is queued using the “queue” keyword, or the job is already running in an NQS batch job, the default is “notify=no”.

Example: `nxnr nastran example notify=yes`

A message is sent when the job completes.

nrec nrec=*number*

nrec=m Where m is the number of external partitions.

nrec is needed for the Recursive Domain Lanczos Method (RDMODES). RDMODES is activated by the nastran keywords dmp and nrec:

NASTRAN dmp=p nrec=m (optional: nclust=c rdscale=x.x)

See the NX Nastran Parallel Processing Guide for more information.

numseg numseg=*number* Default: See text.

Sets the number of frequency segments for either the fdmodes parallel method or the multisegment serial method. See the “EIGRL” bulk entry in the *NX Nastran Quick Reference Guide*.

Note: In a DMP job, the default is the number of tasks specified by the "dmparallel" keyword.

old old=yes,no Default: Yes

Saves previous copies of the F04, F06, LOG, OP2, OUT, PCH, and PLT output files using sequence numbers (additional user-specified file types can be versioned with the "oldtypes" keyword). Sequence numbers are appended to the keyword filename and are separated by a period.

If "yes" is specified, the highest sequence number of each of the output files is determined. The highest sequence number found is incremented by one to become the new sequence number. Then, all current output files that do not include sequence numbers are renamed using the new sequence number as a type.

Example: `nxnr nastran example old=yes`

For example, assume your current working directory contains the following files:

```
v2401.dat v2401.f04.1 v2401.f06 v2401.log
v2401.log.1 v2401.f04 v2401.f04.2 v2401.f06.1
v2401.log.1 v2401.log.3
```

Apparently, the user ran the job four times, but deleted some of the files, e.g., v2401.f04.3, v2401.f06.2, and v2401.f06.3. When the job is run again with "old=yes", the files are renamed as follows: v2401.f04 is renamed to v2401.f04.4, v2401.f06 is renamed to v2401.f06.4, and v2401.log is renamed to v2401.log.4. The sequence number 4 is used because it is one greater than the highest sequence number of all of the selected files (the highest being v2401.log.3).

oldtypes oldtypes=*list* Default: None

Specifies additional file types that will be subject to versioning and deletion via the "old" keyword. The items in the list may be separated by either spaces or commas; they should not include the leading ".". You may specify file types that do not exist.

Example: `nxnr nastran example oldtypes=xdb,mytype`

The files "example.xdb" and "example.mytype" will be subject to versioning or deletion as specified by the "old" keyword.

This keyword may also be set by the NXN_OLDTYPES environment variable. The environment variable overrides the RC files, and the command line overrides the environment variable.

out out=*pathname* Default: .

Saves the output files using a different file prefix or in a different directory. If "out" is not specified, the output files are saved in the current directory using the basename of the input data file as a prefix. If the "out" value is a directory, output files are created in the specified directory using the basename of the input data file as the filename.

In the following examples, assume the current directory includes sub-directories "mydir" and "other", and that an "example.dat" exists in both the current directory and "other". That is, ./example.dat, ./mydir, ./other, and ./other/example.dat exist on Linux; and .\example.dat, .\mydir, .\other, and .\other\example.dat exist on Windows.

Example: `nxnr nastran example`

or: `nxnr nastran other/example`

Output files are created in the current directory with the name “example”, e.g., ./example.f06 on Linux and .\example.f06 on Windows.

Example: `nxnr nastran example out=myfile`

Output files are created in the current directory with the name “myfile”, e.g., ./myfile.f06 on Linux and .\myfile.f06 on Windows.

Example: `nxnr nastran example out=mydir`

Output files are created in the mydir directory with the name “example”, e.g., ./mydir/example.f06 on Linux and .\mydir\example.f06 on Windows.

Example: `nxnr nastran example out=mydir/myfile`

Output files are created in the mydir directory with the name “myfile”, e.g., ./mydir/myfile.f06 on Linux and .\mydir\myfile.f06 on Windows.

parallel parallel=*value* Default: 0

Specifies the maximum number of CPUs selected for shared-memory parallel (SMP) processing in several numeric modules. SMP processing reduces elapsed time at the expense of increased CPU time. The default is 0, which specifies no SMP processing. If “parallel=1", the parallel algorithms are used on one processor.

Note: If you need to vary the number of SMP CPUs during a job, you must set either the “parallel” keyword or SYSTEM(107) on a NASTRAN statement to the maximum number of SMP CPUs that will be requested. Some systems cannot process a DMAP request for CPUs in excess of this initial value.

Example: `nxnr nastran example parallel=2`

The job is run in SMP mode on a maximum of two CPUs.

pause pause=keyword Default: No

Pause the nastran command before exiting to wait for the “Enter” or “Return” key to be pressed. This can be useful when the nastran command is embedded within another program. The values are “fatal”, “information”, “warning”, “yes”, and “no”. Setting “pause=yes” will unconditionally wait; “pause=fatal”, “pause=warning”, and “pause=information” will only wait if a fatal, warning, or information message has been issued by the nastran command. The default is “pause=no”, i.e., do not wait when the nastran command ends.

post post=*command_string* Default: None

Runs the specified command after the job has completed and after the F06, F04, and LOG files have been concatenated if “append=yes” is specified. For Linux, the command must be a valid Korn shell command. The command may pipe the output of one command into another. If the specified command contains embedded spaces, enclose the entire *command_string* in quotes. Each occurrence of the “post” keyword will be concatenated together to form a sequence of commands. Specify a null value, i.e., “post=” to erase all of the previously entered commands. Typical uses of this keyword are to run postprocessing programs or to compress the output files to save space.

Linux example: `nxnr nastran example post='gzip example*'`

At the end of the job, the command “gzip example*” is run to compress all files beginning with “example”.

The value of the “out” keyword is available for use by the “post” keyword. The example “post” keyword could also have been written as post=’gzip $NXN_OUT.*’. If app=yes was specified, post=’gzip $NXN_OUT.out’ would only compress the output file.

Windows example: `nxnr nastran example post=”print example.*”`

At the end of the job, all files named “example.*” will be printed. The output of the post command(s) will be displayed on the command shell window.

See the Environment Variable Table section for a list of environment variables that may be used in the post command.

Note: To allow the “post” keyword to operate on the output files, the standard output from the post commands is not written to the output files.

ppcdelta

(Linux)

ppcdelta=*time* Default: None

Note: The following capability is dependent upon the queue submission commands defined by the “submit” keyword and your queuing system. The capability or examples may not work on your system.

Specifies the amount of time to subtract from the specified CPU time to determine the per-process CPU time limit. This subtraction will ensure that NX Nastran does not consume all of the time allocated to the job.

The value can be specified as either “hours:minutes:seconds”, “minutes:seconds”, or “seconds”, and will always be converted to the number of seconds.

Example:

```
nxnr nastran example \
queue=small cpu=1000 ppcdelta=5
```

The job is submitted to the small queue with a total CPU time limit of 1000 seconds; the NX NASTRAN job will be limited to 995 seconds.

ppmdelta

(Linux)

ppmdelta=*memory_size* Default: 105% of executable size

Note: The following capability is dependent upon the queue submission commands defined by the “submit” keyword and your queuing system. The capability or examples may not work on your system.

Specifies the amount of memory to add to the “memory” value to determine “ppm”, the per-process memory value. The per-process limit is the total amount of memory that each process may acquire. This includes the executable, open core memory (via the “memory” keyword), disk file buffers, and etc..

The *size* is specified as a memory size, see “Specifying Memory Sizes”.

If *size* is less than 1000, then “ppmdelta” equals size divided by 100 and multiplied by the size of the executable, i.e., 105 specifies the default 105% of executable size.

If *size* is greater than 1000, but less than the size of the executable, then “ppmdelta” equals size plus the executable size.

If *size* exceeds the size of the executable, then “ppmdelta” equals size.

Example:

```
nxnr nastran example \
queue=small mem=100m ppmdelta=10m
```

The job is submitted to the small queue with a open core size of 100 MW, and a per-process memory limit of 110 MW.

pre

pre=*command* Default: None

Runs the specified command before the job begins. For Linux, the command must be a valid Korn shell command. The command may pipe the output from one command to another. If the specified command contains embedded spaces, enclose the entire command in quotes. Each occurrence of the “pre” keyword will be concatenated together to form a sequence of commands. Specify a null value, i.e., “pre=” to erase all of the previously entered commands.

Note: The following example may not work on your system.

Linux example:

```
nxnr nastran example \
pre="print Job beginning |\
mail $(whoami)"
```

Sends mail to the submitting user immediately before beginning the job.

Windows example: `nxnr nastran example pre=”dir example.*”`

At the end of the job, a directory listing of all files named “example.*” will be displayed in the LOG file.

See “Environment Variables”, for a list of environment variables that may be used in a “pre” command.

prmdelta

(Linux)

prmdelta=*size* Default: 5120

Note: The following capability is dependent upon the queue submission commands defined by the “submit” keyword and your queuing system. The capability or examples may not work on your system.

Specifies the amount of memory to add to the specified “ppm” value to determine “prm”, the per-request or per-job memory value. The per-job limit is the total amount of memory that all processes in the job may acquire. This includes the NX Nastran process plus any other concurrent or parent processes. The minimum value is 5120.

The *size* is specified as a memory size, see “Specifying Memory Sizes”.

Example: `nxnr nastran example \ queue=small prmdelta=10k`

The per-job memory limit is 10 KW larger than the per-process memory limit.

processor

processor=*file_type* Default: Computer dependent

Specifies the file type of the solver executable. On some computers, NX Nastran provides more than one executable. The baseline executable has the filename “analysis” on Linux and is “analysis.exe” on Windows. Other, advanced-architecture executables are named “analysis.*file_type*” on Linux and “analysis.*file_type*.exe” on Windows. The nastran command will select the correct executable based on the current computer. In some cases, it may be desirable to use one of the other executables. For example, to run the baseline executable on an advanced system, specify “proc=”. To run an advanced-architecture on a new computer not correctly identified by the nastran command, specify “proc=*file_type*”.

Note: This keyword overrides the processor selection logic. Specification of an incompatible executable may cause errors or incorrect operations.

qclass (Linux)

qclass=*string* Default: None

Note: The following capability is dependent upon the queue submission commands defined by the “submit” keyword and your queuing system. The capability or examples may not work on your system.

qoption (Linux)

qoption=*string* Default: None

Note: The following capability is dependent upon the queue submission commands defined by the “submit” keyword and your queuing system. The capability or examples may not work on your system.

Defines the options to add to the queue submittal command. See the “submit” keyword.

Example: `nxnr nastran example \ queue=small qoption=-mu`

The job is run with the additional job submission parameter “-mu” if the keyword reference %qopt% was included in the queue’s command definition.

queue (Linux)

queue=*string* Default: None

Note: The following capability is dependent upon the queue submission commands defined by the “submit” keyword and your queuing system. The capability or examples may not work on your system.

Specifies the name of the queue to use for job submittal. This keyword requires the submit keyword to define the available queues and queue submittal commands. See the “submit” keyword.

Example: `nxnr nastran example queue=small`

This example submits the job to the small queue.

rank

rank=*number* Default: See “System Descriptions”

Sets both SYSTEM(198) and SYSTEM(205) to the specified value. SYSTEM(198) and SYSTEM(205) set the minimum front size and number of rows that are simultaneously updated, respectively, in sparse symmetric decomposition and FBS. The sparse solver will build a front, a k k sub matrix, until k is at least as large as SYSTEM(198). Once a sufficiently large front has been built, it is updated m rows at a time, where m is the value of SYSTEM(205).

For best performance, *SYSTEM (205) ≥SYSTEM (198)* . The optimal values for these system cells is problem and processor dependent; the default values for these system cells are set to processor-dependent values.

The actual value used for SYSTEM(205) may be found in the F04 file in the text of USER INFORMATION MESSAGE 4157 as the RANK OF UPDATE value. See Table C-3 for the default values of these system cells.

rdscale

rdscale=*number*

rdscale is used by the Recursive Domain Lanczos Method (RDMODES), and is a factor to modify the selected frequency range in the EIGRL specification for eigensolutions of each substructure. The default value of 1.0 uses the same frequency range as EIGRL specification.

See the NX Nastran Parallel Processing Guide for more information.

real real=*size* Default: See text.

Specifies the amount of open core memory that certain numerical modules will be restricted to. This keyword may be used to reduce paging, at the potential expense of spilling. The keyword may also be set with the “sys81" keyword. See the *NX Nastran Quick Reference Guide* for further information.

The *size* is specified as a memory size, see “Specifying Memory Sizes”.

On Linux systems, the default is “0”. On Windows systems, the default is calculated using “realdelta”.

realdelta realdelta=*size* Default: 12MB

Specifies the difference between physical memory and the “real” parameter if neither “real” nor “sys81” were set.

The *size* is specified as a memory size, see “Specifying Memory Sizes”.

(Windows)

If *size* is greater than 1000, the value is subtracted from the physical memory size.

If *size* is less than 1000, it is assumed to be a percentage of the physical memory size.

Example: `nxnr nastran example realdelta=50`

The “real” value will be set to 50% of the physical memory if no value has been assigned to “real” or SYSTEM(81).

resd resd=yes,no Default: Yes

This keyword may also be set by the MP_RESD environment variable. The environment variable overrides the RC files, and the command line overrides the environment variable.

rcf rcf=*pathname* Default: None

Specifies the name of the local RC file. If this keyword is not specified, the .nast1rc file on Linux and nastr.rcf on Windows located in the input data file’s directory is used.

Example: `nxnr nastran example rcf=nast.rcf`

The nastran command will process ./nast.rcf on Linux, or .\nast.rcf on Windows in lieu of the default local RC file ./.nastrrc on Linux and .\nastr.rcf on Windows.

rcmd rcmd=*pathname* Default: See text.

(Linux)

Specifies the path of the nastran command on the remote system when remote processing has been requested via the “node” keyword. If this value is not set, the nastran command will first try its own absolute path on the remote system, if this fails, the path will be removed, i.e., the default PATH of the remote system will be used.

Example: `nxnr nastran example \ rcmd=/siemens/bin/nxnr`

The pathname of the nastran command on the remote system is explicitly defined as /*install_dir*/bin/nxn*r*. If this file does not exist, or is otherwise not executable, the job will fail.

rmpool

rmpool=*number* Default: See your System Administrator.

This keyword may be set with the MP_RMPOOL environment variable. The environment variable overrides the RC files; the command line overrides the environment variable.

scr300co

scr300co=*value* Default: 1

Allows you to define a factor to scale SCR300 estimates. This scale factor is applied before the "scr300min" value, that provides a lower bound for SCR300 estimates.

Example: `nxnr estimate example scr300co=2`

This will double the SCR300 disk estimate and then apply the "scr300min" lower bound.

Example: `nxnr estimate example scr300co=0.5`

This will halve the SCR300 disk estimate. An estimate less than the lower bound specified by "scr300min" will be set to the lower bound.

scr300min

scr300min=*value* Default: 1mb

Allows you to define the lower bound for all SCR300 estimates. This bound is applied after the "scr300co" value, that multiplies the actual estimate by a "conservatism" factor.

Example: `nxnr estimate example scr300min=2mb`

This will set the minimum SCR300 disk estimate to 2 MB.

scratch

scratch=yes,no,mini Default: No

Deletes the database files at the end of the run. If the database files are not required, “scratch=yes” can be used to remove them preventing cluttering of the directory with unwanted files. If “mini” is specified, a reduced size database that can only be used for data recovery restarts will be created. See Chapter 16 of the NX Nastran Users Guide for further details on the “mini” database.

Example: `nxnr nastran example scratch=yes`

All database files created by the run are deleted at the end of the job in the same way as the FMS statement INIT MASTER(S).

sdball

sdball=*size* Default: Computer dependent

Specifies an alternate default size of the DBALL DBSet. The computer-dependent default is listed in “Computer Dependent Defaults”. This default is overridden by an INIT FMS statement. If the value “sdball=estimate” is specified, ESTIMATE will be used to determine a suitable default.

The size is specified as the number of blocks (BUFFSIZE words long) or the number of words follwed by one of the modifiers: “G”, “GW”, “GB”, “M”, “MW”, “MB”, “K”, “KW”, “KB”, “W”, “B”. See “Specifying Memory Sizes” for a description of these modifiers.

Note: The software doesn't verify whether the DBALL DBSet could ever grow to the size specified by this keyword.

Example: `nxnr nastran example sdball=1024gb`

Defines the default size of the DBALL DBSet as 1 TB.

sdirectory sdirectory=*directory* Default: See description below.

See "Determining Resource Requirements" for information on estimating a job's total disk space requirements.

Specifies the directory to use for temporary scratch files created during the run. NX Nastran can create very large scratch files, the scratch directory should contain sufficient space to store any scratch files created during a run. You must have read, write, and execute privileges to the directory.

Linux: The default value is taken from the TMPDIR environment variable if it is set to a non-null value. Otherwise the computer's default temporary file directory is chosen; this is usually /tmp.

Windows: The default value is taken from the TEMP environment variable.

Linux example: `nxnr nastran example sdir=/scratch`

Scratch files are created in the /scratch directory.

Windows example: `nxnr nastran example sdir=d:\scratch`

Scratch files are created in the d:\scratch directory

If a DMP run was selected with *dmparallel* ≥*1* , unique task-specific scratch directories may be set for each host using the standard PATH separator, i.e, ":" on Linux and ";" on Windows, to separate entries. The directories will be paired with each host in a round-robin order, that is, the list will be reused if more tasks than directories are specified.

Linux example: `nxnr nastran example dmp=4 \ sdir=/scratch1:/scratch2`

In this example, /scratch1 will be used for the first and third tasks, while /scratch2 will be used for the second and fourth tasks.

slavejob slavejob=yes,no Default: No

Specifies a special "slave job" is to be run on the slave nodes.

slaveout slaveout=yes,no Default: No

Specifies the output files from the slave nodes are to be copied back to the local node.

smaster smaster=*size* Default: Computer dependent

Specifies an alternate default size of the MASTER DBSet. The computer-dependent default is listed in "Computer Dependent Defaults". This default is overridden by an INIT FMS statement.

The size is specified as the number of blocks (BUFFSIZE words long) or the number of words follwed by one of the modifiers: "G", "GW", "GB", "M", "MW", "MB", "K", "KW", "KB", "W", "B". See "Specifying Memory Sizes" for a description of these modifiers.

Note: The software doesn't verify whether the MASTER DBSet could ever grow to the size specified by this keyword.

Example: `nxnr nastran example smaster=1024gb`

Defines the default size of the MASTER DBSet as 1 TB.

smemory smemory=*value* Default: 100

Specifies the memory to reserve for scratch memory. The keyword smemory can be specified either as the number of GINO blocks (default), or as a number followed by one of the following modifiers:

Note

This keyword is overridden by the FMS statement INIT SCRATCH(MEM=*value*).

G or Gw	Multiply *entered_value* by 1024**3.
Gb	Multiply *entered_value* by (1024**3)/*bytes_per_word*.
M or Mw	Multiply *entered_value* by 1024**2.
Mb	Multiply *entered_value* by (1024**2)/*bytes_per_word*.
K or Kw	Multiply *entered_value* by 1024.
Kb	Multiply *entered_value* by 1024/*bytes_per_word*.
w	Use *entered_value* as is.
b	Divide *entered_value* by *bytes_per_word*.

where *bytes_per_word* is 4 using the LP-64 executables, and 8 using ILP-64 executables. The modifier may be specified using any case combination.

Note

NX Nastran uses standard computer units for K, M, and G.

Example: `nxnr nastran example smem=200`

This example reserves 200 GINO blocks for scratch memory.

sparse sparse=*number* Default: See the *NX Nastran Quick Reference Guide*.

Sparse matrix method selection. This keyword may also be set with the “sys126" command line keyword. See the *NX Nastran Quick Reference Guide* for information on the default value and legal values for this keyword.

sscr sscr=*size* Default: Computer dependent

Specifies an alternate default size of the SCRATCH DBSet. The computer-dependent default is listed in “Computer Dependent Defaults”. This default is overridden by an INIT FMS statement. If the value “sscr=estimate” is specified, ESTIMATE will be used to determine a suitable default.

The size is specified as the number of blocks (BUFFSIZE words long) or the number of words follwed by one of the modifiers: “G”, “GW”, “GB”, “M”, “MW”, “MB”, “K”, “KW”, “KB”, “W”, “B”. See “Specifying Memory Sizes” for a description of these modifiers.

Note: The software doesn't verify whether the SCRATCH DBSet could ever grow to the size specified by this keyword.

Example: `nxnr nastran example sscr=1024gb`

Defines the default size of the SCRATCH DBSet as 1 TB.

submit submit=[*list*=]*definition* Default: None

(Linux)

Defines the command and options used to run a job when the “queue” keyword is specified. The “submit” keyword, only specified in RC files, consists of an optional queue list, followed by the command definition for the specified queues as shown below:

```
submit=list=command
 submit=command
```

When specified, the list contains one or more “queue” names separated by commas. If a queue list is not supplied, the command applies to all queues.

The *command* section of the “submit” keyword value defines the *command* used to run a job when a “queue” keyword is supplied that matches a queue name in the *list*. The *command* can contain keyword names enclosed in percent “%” signs that are replaced with the value of the keyword before the *command* is run. A complete description of the command is found in “Customizing Queue Commands (Linux)”.

symbol symbol=*name*=*string* Default: None

Defines a symbolic (or logical) name used on ASSIGN and INCLUDE statements and in command line arguments. This statement can only be specified in initialization or RC files. It *cannot* be specified on the command line (although logical symbols defined using this keyword may be used on the command line). The symbol definition can include references to previously defined symbols or environment variables use the standard “$*name*” or “${*name*}” syntax on Linux, or “%*name*%” on Windows.

When the keywords “hosts” or “node” are specified, symbolic names defined using the symbol keyword are passed to the remote systems and not used by the local system. The path names must be valid on the remote systems. In this case, the lsymbol keyword can be used instead of symbol to specify symbolic names for the local system.

When neither “hosts” or “node” are specified, symbolic names defined with the symbol keyword are treated as local. In this case, symbolic names defined using the lsymbol keyword will over-ride symbolic names specified using the symbol keyword.

Symbolic names must be 16 characters or less, the value assigned to the symbolic name must be 256 characters or less. If the symbolic name used in ASSIGN or INCLUDE statements or in command line arguments is not defined, it is left in the filename specification as is.

For example, many of the TPL and DEMO input data files have ASSIGN statements, such as the following:

```
ASSIGN 'MASTER=DBSDIR:abc.master'
```

The string “DBSDIR:” specifies a symbolic name that is to be replaced by another string. The replaced string is defined by the “symbol” keyword in the initialization or RC file or as an environment variable. For example,

```
Linux: SYMBOL=DBSDIR=/dbs
 Windows: SYMBOL=DBSDIR=d:\dbs
```

When the previous ASSIGN statement is processed, the filename assigned to the logical name MASTER is /dbs/abc.master on Linux and d:\dbs\abc.master on Windows. An alternate way of defining symbolic names is through the use of environment variables. For example, typing the following command at a Korn shell prompt

```
export DBSDIR=/dbs
```

at a Korn shell prompt,

```
setenv DBSDIR /dbs
```

at a C-shell prompt, or

```
set DBSDIR=d:\dbs
```

at a Windows shell prompt, is equivalent to the above “symbol” keyword definitions.

Note: If a symbolic name is defined by both an RC file and an environment variable, the symbol statement value will be used.

The section titled “Environment Variables” contains a list of environment variables that are automatically created by the nastran command. Of particular importance to the logical symbol feature are the OUTDIR and DBSDIR variables. These variables refer to the directory that will contain the output files (set using the “out” keyword) and the directory that will contain the permanent database files (set using the “dbs” keyword), respectively.

sysfield

sysfield=*string* Default: None

Defines a global SYS value that is applied to all DBsets. See the sections titled “Using the SYS Field” or “SYS Parameter Keywords” for further details.

Example: `nxnr nastran example sysfield=lock=no`

This example disables file locking for all DBsets.

sysn

sysn=*value* Default: None

Sets the SYSTEM(n) to value. This keyword may be repeated any number of times. All non repeated cells are used, but only the last repeated cell is used. The form “system(n)=value”, may also be used, but the entire keyword-value string must be quoted when used on a Linux command line.

Example:

```
nxnr nastran example sys2=19
```

or

```
nxnr nastran example "system(2)=19"
```

These examples set SYSTEM(2) to 19. The second example shows how to quote the parenthetic form.

threads

threads=*value* Default: None

Threads are used to implement NX Nastran tasks. For maximal performance, there should be one thread per NX Nastran task and one processor per thread. An excess number of threads will not help performance; if there are more NX Nastran tasks than threads or more threads than processors, a longer elapsed time will result.

The Dynamic Thread Management feature is available only in the NX Nastran Rank-N sparse solver (see the “rank” keyword). The Rank-N sparse solver is used widely in linear static analysis and Lanczos eigenvalue analysis jobs. Other NX Nastran parallel modules will run with a constant number of threads specified by the PARALLEL keyword

The “threads” keyword specifies the suggested number of threads to be maintained by the Dynamic Thread Management feature. Setting a value for “threads” causes the runtime library to create an additional asynchronous “monitor” process that periodically awakens to monitor system load. When idle processors exist, this monitor process increases the number of threads up to the maximum that is specified by the “parallel” keyword. As the system load increases, the monitor process decreases the number of threads, possibly to as few as one. If “threads” has not been set, this feature is disabled and the constant number of threads specified via the “parallel” keyword will be used.

This keyword may also be set by the MP_SUGNUMTHD environment variable. The environment variable overrides the RC files, and the command line overrides the environment variable.

thread_max thread_max=*value* Default: parallel

Specifies an upper bound on the number of threads that a job will use when “threads” is also set. The value must satisfy the relation $thread_{min} \leq thread_{max} \leq parallel$, where *parallel* is the value specified by the “parallel” keyword.

This keyword may also be set by the MP_SUGNUMTHD_MAX environment variable. The environment variable overrides the RC files, and the command line overrides the environment variable.

thread_min thread_min=*value* Default: 1

Specifies a lower bound on the number of threads a job will use when “threads” is also set. The value must satisfy the relation $1 \leq thread_{min} \leq thread_{max}$.

This keyword may also be set by the MP_SUGNUMTHD_MIN environment variable. The environment variable overrides the RC files, the command line overrides the environment variable.

thread_verbose thread_verbose=yes,no Default: No

Controls the output of informational messages. If “thread_verbose=yes” is set, the monitoring process will write messages to the LOG file whenever it changes the number of threads.

This keyword may also be set by the MP_SUGNUMTHD_VERBOSE environment variable. The environment variable overrides the RC files, the command line overrides the environment variable.

trans trans=yes,no,auto Default: no (local); auto (remote)

If the “node” keyword is not specified, this keyword indicates the XDB file is to be translated to a neutral-format file using the TRANS utility. The output file will have the file type “.ndb”.

Linux only: If the “node” keyword is specified, this keyword indicates how an XDB file is to be copied back to the local node. If “trans=auto” is specified, the XDB file will be copied using TRANS/RECEIVE if the two computers use different floating point formats or by a binary copy if the floating point formats are the same. If “trans=yes” is specified, the XDB is always copied using TRANS on the remote node and RECEIVE on the local node (this may be needed if the floating point formats are identical but the file formats are not). If “trans=no” is specified, the XDB file will not be copied back

Example: `nxnr nastran example trans=yes`

This example will run NX Nastran and then convert the XDB file, if written, to neutral format using TRANS.

Linux example: `nxnr nastran example node=othernode \ trans=yes`

This example will run NX Nastran on node othernode and copy the XDB file back using TRANS/RECEIVE.

username (Linux)

username=*name* Default: Current user name

Specifies an alternate username on the remote host when the “node” keyword is specified. This keyword may only be specified on the command line.

Example:

```
nxnr nastran example node=othernode \
 user=fred
```

This example will run NX Nastran on node othernode as user “fred”.

usparse

usparse=*number* Default: See the description below.

Unsymmetrix sparse matrix method selection. This keyword may also be set with the “sys209" command line keyword. See the *NX Nastran Quick Reference Guide* for information on the default value and legal values for this keyword.

version

version=*version_number* Default: *Latest installed version*

Specifies the version number. The keyword may only be specified on the command line or in the command initialization file.

Example: `nxnr nastran example version=1.0`

This example will run NX Nastran version 1.0 assuming it has been installed in the same installation base directory as this version of the product.

xhost (Linux)

xhost=yes,no Default: No

Indicates if the xhost(1) command is to be run. The xhost(1) command may be required if the “node” keyword and either “xmon=yes” or “xmon=kill” are specified. The argument to xhost(1) will be the node specified by the “node” keyword. This keyword is ignored if the “node” keyword is not specified.

B.1 SYS Parameter Keywords

buffio buffio=yes,no,must Default: No

(See Table 5-7)

This keyword specifies the file is to be buffered. If “buffio=yes” is specified and a memory allocation operation fails, then unbuffered disk I/O will be used. If “buffio=must” is specified and a memory allocation operation fails, then a fatal error will be issued and the job terminated. See “Using Buffered I/O” for further information.

lock

(Linux)

lock=yes,no Default: No for Delivery DBsets

Yes for all others.

Specifies the file is to be locked when it is opened. Locking a file prevents two or more NX Nastran jobs from interfering with one another; however, this does not prevent any other program or operating system command from modifying the file.

SYSTEM(207) can also be used to globally control DBset locking. Setting SYSTEM(207)=1 will disable locking unless overridden for a specific file by SYS=LOCK=YES on an ASSIGN FMS statement. Setting SYSTEM(207)=0 will enable locking of read-write DBsets unless overridden for a specific file by SYS=LOCK=NO on an ASSIGN FMS statement.

mapio

mapio=yes,no,must Default: No

(See Table 5-7)

This keyword specifies the file is to be mapped. If “mapio=yes” is specified and a mapping operation fails, then normal disk I/O will be used. If “mapio=must” is specified and a mapping operation fails, then a fatal error will be issued and the job terminated. See “Using File Mapping” for further information.

wnum

wnum=*number* Default: 4

(See Table 5-7)

Specifies the number of windows or buffers that will be maintained for each mapped or buffered file. The use of multiple windows or buffers permits multiple I/O streams to target a file (e.g., simultaneously reading one matrix and writing another) without forcing an excessive number of window remap operations or buffered read/writes. The number must be between 1 through 16 inclusive, values outside of this range are ignored without acknowledgement.

wsize

wsize=*size* Default: The larger of 128KB or 4*BUFFSIZE.

(See Table 5-7)

File Mapping. Specifies the size of the window mapping the file into memory. The window is that portion of the file that is visible through the map. If the window is the same size as the file, then the entire file is visible. If the window is smaller than the file, then any portion of the file within the window or windows can be directly accessed; the rest of the file cannot be accessed until a window is remapped to include the desired file location.

(See Table 5-7)

Buffered I/O. Specifies the size of the buffer read from or written to disk. If the buffer is the same size as the file, then the entire file is memory resident. If the buffer is smaller than the file, then any portion of the file within the buffer or buffers can be directly accessed; the rest of the file cannot be accessed until a buffer is read to include the desired file location.

The total window or buffer size (WNUM value * WSIZE value) is limited to 25% of the available address space or, for windows, to 25% of the physical memory. The address space limit is displayed by the "limits" special function, see "Using the Help Facility and Other Special Functions", as the "Virtual Address Space" limit. If "wsize=0" is specified for a read-only file, the entire file will be mapped or buffered into memory, subject to the 25% address space limit.

The *size* is specified as a memory size, see "Specifying Memory Sizes".

If *size* is less than the file's BUFFSIZE, then *size* is multiplied by BUFFSIZE.

B.2 Environment Variables

The following environment variables affect the execution of the nastran command.

Table B-1. Environment Variables Affecting the nastran Command

Name	Purpose
HOME	Linux: The user's home directory.
HOMEDRIVE	Windows: The user's home drive.
HOMEPATH	Windows: The user's home directory.
LM_LICENSE_FILE	Alternate means to set the "authorize" keyword.
LOGNAME	Linux: The user ID.
NXN_ARCH	Specifies the NX Nastran architecture.
NXN_BASE	If set, the script will use this directory as the *install_dir*.
NXN_ISHELLEXT	Alternate means to set the "ishellext" keyword.
NXN_ISHELLPATH	Alternate means to set the "ishellpath" keyword.
NXN_JIDPATH	Alternate means to set the "jidpath" keyword.
NXN_LICENSE_FILE	Alternate means to set the "authorize" keyword.
NXN_NOEXE	If set, the nastran command will build the execution script but will not actually execute it. This may be useful for debugging purposes.
NXN_OLDTYPES	Alternate means to set the "oldtypes" keyword.
NXN_VERSD	Internal Siemens use only.
NXNDBG	Specify debugging flags.
TEMP	Windows: If set, this is the default value for the "sdirectory" keyword. If not set, use the system default temporary file directory as the default value.
TMPDIR	Linux: If set, this is the default value for the "sdirectory" keyword. If not set, use the system default temporary file directory as the default value.
USER	Linux: The user's home directory (if LOGNAME is not set or is a null string).

The following environmental variables are available for use by the “pre” and “post” keywords.

Table B-2. “Pre” and “Post” Keyword Environment Variables

Name	Purpose
DBSDIR	The directory part of NXN_DBS, i.e., the directory that will contain the permanent database files.
DELDIR	Directory containing the solution sequence source files (*install_dir*/nxn*r*/nast/del on Linux and *install_dir*\nxn*r*\nast/del on Windows).
DEMODIR	Directory containing DEMO library (*install_dir*/nxn*r*/nast/demo on Linux and *install_dir*\nxn*r*\nast\demo on Windows).
JIDDIR	Directory containing the input file.
NXN_APP	yes,no
NXN_ASG	Internal use only.
NXN_ARCH	The actual architecture used by the nastran command.
NXN_LICENSE_FILE	Licensing value.
NXN_BASE	The actual *install_dir* used by the nastran command.
NXN_DBS	Default prefix of permanent databases.
NXN_EXE	Executable path.
NXN_JID	Input data file path.
NXN_MEM	Open core memory size in words.
NXN_OLD	yes,no
NXN_OUT	Prefix of F06, F04, and LOG files.
NXN_SCR	yes,no
NXN_SDIR	Default prefix of scratch databases.
NXN_VERSD	Internal use only.
OUTDIR	Output file directory.
SSSALTERDIR	Directory containing SSS alters (*install_dir*/nxn*r*/nast/misc/sssalter on Linux and *install_dir*\nxn*r*\nast\misc\sssalter on Windows).
TEMP	Windows: Temporary directory.
TMPDIR	Linux: Temporary directory.
TPLDIR	Directory containing TPL library (*install_dir*/nxn*r*/nast/tpl on Linux and *install_dir*\nxn*r*\nast\tpl on Windows).

B.3 Other Keywords

The following keywords are available for use by the nastran command and script templates. You will generally not need to set or use these values.

Table B-3. Other Keywords

Keyword	Purpose
0	Pathname of the nastran command.
0.acceptdeny	Pathname of accept/deny utility used in this job.
0.dmp	DMP job template pathname.

0.dmpaccept	Pathname of dmpaccept utility.
0.dmpdeny	Pathname of dmpdeny utility.
0.ini	Command initialization file pathname.
0.lcl	Local job template pathname.
0.rmt	Remote job template pathname.
0.rmtaccept	Pathname of rmtaccept utility.
0.rmtdeny	Pathname of rmtdeny utility.
0.srv	Server job template pathname.
0.tmplt	Alternate template pathname, overrides local/remote template selection logic.
a.appdir	Application specific base pathname relative to NXN_BASE.
a.archdir	Architecture specific base pathname relative to NXN_BASE.
a.estimate	ESTIMATE executable filename relative to “a.archdir”.
a.flex	Pathname of default FLEXlm license file.
a.fms	Comma-separated list of FMS keywords recognized in RC files.
a.k	Multiplier for K factor.
a.msgcat	Pathname of default message catalog.
a.news	News filename relative to “a.appdir”.
a.port	Default FLEXlm port number.
a.rc	RC file basename. User RC files are prefixed by “.”.
a.receive	RECEIVE executable filename relative to “a.archdir”.
a.release	Release number, same as NX Nastran version number.
a.sbcm	Pathname of default node-locked authorization code file.
a.solver	Solver executable filename relative to “a.archdir”.
a.sss	Delivery database filename relative to “a.archdir”.
a.tier	Reserved internal variable.
a.touch	News file touch pathname.
a.trans	TRANS executable filename relative to “a.archdir”.
a.urc	File name of default User RC file.
d.hosts	Blank separated list of per-task hostnames
d.jidvis	Blank separated list of per-task JID visibility flags.
d.outvis	Blank separated list of per-task output directory visibility flags.
d.rcmds	Blank separated list of per-task “rcmd” values.
d.sdirs	Blank separated list of per-task “sdirectory” values.
d.tid	DMP task ID.
dcmd	Debugger.
debug	Run solver under debugger.
j.all	Blank separated list of file types to be deleted at job completion if “delete=all” is specified.
j.app	Blank separated list of file types to be appended at job completion if “append=yes” is specified.
j.base	Job basename.
j.command	Job submittal command string.

j.dir	Job directory.
j.env	Job environment variable list.
j.msg	Job completion message.
j.nascar	List of NASTRAN entries.
j.news	News file pathname.
j.ofp	Blank separated list of file types to be deleted at job completion if and only if they are empty.
j.old	Blank separated list of file types to be versioned or deleted under the “old” keyword.
j.out	Appended output file type.
j.rcfiles	Comma-separated list of RC files.
j.server	NX Nastran server flag
j.shell	Shell debugging flag.
j.startdate	Job start date-time string.
j.tty	TTY name.
j.type	Space separated list of file types to be versioned.
j.unique	Job unique name.
job	Job script filename, created in out directory.
log	Pathname of LOG file.
msgdest	System message destination.
nprocessors	Number of processors.
ppc	Per-process CPU time limit.
ppm	Per-process memory limit.
prm	Per-request memory limit.
PWD	Current working directory.
r.argv	List of arguments to be processed on rmt/dmp host.
r.jidvis	JID visibility flag.
r.outvis	Output directory visbility flag.
s.arch	System architecture name.
s.block	Words per disk block.
s.bpw	Bytes per word.
s.clock	CPU clock frequency.
s.config	CONFIG number.
s.cpu	CPU name.
s.hostname	Simple hostname.
s.model	System model name.
s.modeldata	Pathname of site specific model data.
s.nproc	Number of processors.
s.numeric	Encoded numerical format.
s.os	OS name.
s.osv	OS version.
s.pmem	Physical memory, in MB. Only known on Windows.
s.proc	Default processor subtype.

s.rawid	Raw configuration number.
s.rsh	Remote shell command.
s.type	System description.
s.vmem	Virtual memory, in MB. Only known on Windows.
tcmd	Timing command.

Appendix C: System Descriptions

C.1 Overview

This section presents quantitative information useful for evaluating the processing requirements of NX Nastran. It includes system descriptions, numerical data, and information on computer dependent defaults.

C.2 System Description Summary

The list of supported systems is included in the README.txt file located with the NX Nastran installation.

C.3 Numerical Data

Table C-1. Numerical Data – 32-bit, big and little endian, IEEE, big endian is read left to right, little endian is read right to left

Item	Description
INTEGER Bit Representation	31 30 … 0 S | Integer
REAL Bit Representation	31 30 23 22 … 0 S | Exponent | Mantissa
Exponent Range for a REAL Number	±38
Precision of a REAL Variable	6 digits (24 bits)
DOUBLE PRECISION Bit Representation	63 62 52 51 … 32 S | Exponent | Mantissa 31 … 0 Mantissa (cont.)
Exponent for a DOUBLE PRECISION Number	±308
Precision of a DOUBLE PRECISION Variable	15 digits (53 bits)

IEEE Standard 754 doesn't define a 128-bit floating point value; the format varies among computer manufacturers.

Table C-2. Numerical Data – 64-bit, big and little endian, IEEE, big endian is read left to right, little endian is read right to left

Item	Description
INTEGER Bit Representation	63 62 … 0 S | Integer

REAL Bit Representation	63 62 52 51 0 S Exponent Mantissa
Exponent Range for a REAL Number	±308
Precision of a REAL Variable	15 digits (53 bits)
DOUBLE PRECISION Bit Representation	Not applicable
Exponent for a DOUBLE PRECISION Number	Not applicable
Precision of a DOUBLE PRECISION Variable	Not applicable

C.4 Computer Dependent Defaults

These tables list the computer-dependent default values for NX Nastran. The default rank values are listed in Table C-3.

Table C-3. Computer-Dependent Defaults

Parameter	Input File Settings	Command Line Settings	Default	Comment
BUFFPOOL	`NASTRAN BUFFPOOL=n`	`bpool=n`	37	GINO Blocks
BUFFSIZE	`NASTRAN BUFFSIZE=n`	`buffsize=n`	8193	Max: 65537
BUFFSIZE Increment	`NASTRAN SYSTEM(136)=n`	`sys136=n`	128	Words
DBALL Size	`INIT DBALL , LOGICAL=(DBALL(n))`	`sdball=n`	1000000	GINO Blocks
DBS Update Time	`NASTRAN SYSTEM(128)=n`	`sys128=n`	5	
Lanczos HPO	`NASTRAN SYSTEM(193)=n`	`sys193=n`	0	Save
Lanczos HPO	`NASTRAN SYSTEM(194)=n`	`sys194=n`	0	Pack/Unpack
SCRATCH Size	`INIT SCRATCH , LOGICAL=(logname(n)), SCR300=(logname(n))`	`sscr=n`	1000000	GINO Blocks
SMEM	`INIT SCRATCH (MEM=n)`	`smem=n`	100	GINO Blocks
Sparse Ordering Method	`NASTRAN SYSTEM(206)=n`	`sys206=n`	4	Prefer Extreme reordering

Table C-4. Computer-Dependent Default Rank Values

Computer Type	Model	SYS198	SYS205
Intel, Linux, Windows	All	6	6

Siemens Industry Software

Headquarters
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 972 987 3000

Americas
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 314 264 8499

Europe
Stephenson House
Sir William Siemens Square
Frimley, Camberley
Surrey, GU16 8QD
+44 (0) 1276 413200

Asia-Pacific
Suites 4301-4302, 43/F
AIA Kowloon Tower, Landmark East
100 How Ming Street
Kwun Tong, Kowloon
Hong Kong
+852 2230 3308

About Siemens PLM Software

Siemens PLM Software, a business unit of the Siemens Industry Automation Division, is a leading global provider of product lifecycle management (PLM) software and services with 7 million licensed seats and 71,000 customers worldwide. Headquartered in Plano, Texas, Siemens PLM Software works collaboratively with companies to deliver open solutions that help them turn more ideas into successful products. For more information on Siemens PLM Software products and services, visit www.siemens.com/plm.