MSC Nastran 2023.1

Installation and Operations Guide

HEXAGON

Americas
5161 California Ave. Suite 200
University Research Park
Irvine, CA 92617
Telephone: (714) 540-8900
Email: americas.contact@hexagon.com

Europe, Middle East, Africa
Am Moosfeld 13
81829 Munich, Germany
Telephone: (49) 89 431 98 70
Email: info.europe@hexagon.com

Japan
KANDA SQUARE 16F
2-2-1 Kanda Nishikicho, Chiyoda-ku
1-Chome, Shinjuku-Ku
Tokyo 101-0054, Japan
Telephone: (81)(3) 6275 0870
Email: MSCJ.Market@hexagon.com

Asia-Pacific
100 Beach Road
#16-05 Shaw Tower
Singapore 189702
Telephone: 65-6272-0082
Email: APAC.Contact@hexagon.com

Worldwide Web
www.hexagon.com

Support
https://simcompanion.hexagon.com

Disclaimer

March 15, 2023

NA:V2023.1:Z:Z:Z:DC-IOG-PDF

Documentation Feedback

At Hexagon Manufacturing Intelligence, we strive to produce the highest quality documentation and welcome your feedback. If you have comments or suggestions about our documentation, write to us.

Please include the following information with your feedback:

- Document name
- Release/Version number
- Chapter/Section name
- Topic title (for Online Help)
- Brief description of the content (for example, incomplete/incorrect information, grammatical errors, information that requires clarification or more details and so on).
- Your suggestions for correcting/improving documentation

You may also provide your feedback about Hexagon Manufacturing Intelligence documentation by taking a short 5-minute survey.

Note:	The above mentioned e-mail address is only for providing documentation specific feedback. If you have any technical problems, issues, or queries, please contact Technical Support.

Contents

3 Configuring MSC Nastran

4 Using the Basic Functions of MSC Nastran

A Configuring the Runtime Environment

B Keywords and Environment Variables

C System Descriptions

Glossary

Preface

List of MSC Nastran Guides

A list of some of the MSC Nastran guides is as follows:

Installation and Release Guides
■ Installation and Operations Guide
■ Release Guide
Reference Guides
■ Quick Reference Guide
■ DMAP Programmer's Guide
■ Reference Guide
■ Utilities Guide
■ Getting Started Guide
■ SOL 400 Getting Started Guide
■ MSC Nastran Error Messages Guide
Demonstration Guides
■ Linear Analysis
■ Implicit Nonlinear (SOL 400)
■ Explicit Nonlinear (SOL 700)
■ MSC Nastran Verification Guide
User's Guides
■ Automated Component Modal Synthesis (ACMS)
■ Access Manual
■ Aeroelastic Analysis
■ Design Sensitivity and Optimization
■ DEMATD
■ Dynamic Analysis
■ Embedded Fatigue
■ Embedded Vibration Fatigue
■ Explicit Nonlinear (SOL 700)
■ High Performance Computing
■ Linear Static Analysis
■ Nonlinear (SOL 400)
■ Numerical Methods
■ Rotordynamics

- Superelements and Modules
- Thermal Analysis
- User Defined Services

You may find any of these documents from Hexagon at:

https://simcompanion.hexagon.com/customers/s/article/MSC-Nastran-Support-Home-Page

Technical Support

For technical support phone numbers and contact information, please visit:
https://simcompanion.hexagon.com/customers/s/article/support-contact-information-kb8019304

Support Center (http://simcompanion.hexagon.com)

The SimCompanion link above gives you access to the wealth of resources for Hexagon products. Here you will find product and support contact information, product documentations, knowledge base articles, product error list, knowledge base articles and SimAcademy Webinars. It is a searchable database which allows you to find articles relevant to your inquiry. Valid MSC customer entitlement and login is required to access the database and documents. It is a single sign-on that gives you access to product documentation for complete list of products from Hexagon, allows you to manage your support cases, and participate in our discussion forums.

Training and Internet Resources

Hexagon corporate site has the information on the latest events, products and services for the CAD/CAE/CAM marketplace.

http://simcompanion.hexagon.com

The SimCompanion link above gives you access to the wealth of resources for Hexagon products. Here you will find product and support contact information, product documentations, knowledge base articles, product error list, knowledge base articles and SimAcademy Webinars. It is a searchable database which allows you to find articles relevant to your inquiry. Valid MSC customer entitlement and login is required to access the database and documents. It is a single sign-on that gives you access to product documentation for complete list of products from Hexagon, allows you to manage your support cases, and participate in our discussion forums.

Design and Engineering e-Learning

The MSC-Training link above will point you to schedule and description of MSC Seminars. Following courses are recommended for beginning Nastran users.

NAS120 - Linear Static Analysis using MSC Nastran and Patran

This seminar introduces basic finite element analysis techniques for linear static, normal modes, and buckling analysis of structures using MSC Nastran and Patran. MSC Nastran data structure, the element library, modeling practices, model validation, and guidelines for efficient solutions are discussed and illustrated with examples and workshops. Patran will be an integral part of the examples and workshops and

will be used to generate and verify illustrative MSC Nastran models, manage analysis submission requests, and visualize results. This seminar provides the foundation required for intermediate and advanced MSC Nastran applications.

1 Introduction

- Document Scope
- Document Structure
- Supported Hardware and Operating Systems
- The Directory Structure

Document Scope

The *MSC Nastran Installation and Operations Guide* (IOG) provides instructions on how to install, customize, and use MSC Nastran 2023.1 on LINUX and Windows systems. This document assumes that you have a working knowledge of the applicable operating environments.

Note:	This document includes information for systems not yet supported by MSC Nastran. Hexagon does not guarantee later support for these systems.

Key for Readers

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

Italics	Represent user-specified variables.	
	Example:	The system RC file is *install_dir*/conf/nast20231rc.
Courier font	Indicates system input or output.	
	Example:	`$ install_dir/bin/msc20231`
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, MSC Nastran saves the output files using the basename of the input data file as a prefix.

Definitions Used in this document

The IOG uses certain definitions to denote installation directories, and product versions of MSC Nastran. The default *install_dir* of MSC Nastran 2023.1 is as follows:

Linux: /msc/MSC_Nastran/2023.1

Windows: C:\Program Files\MSC.Software\MSC_Nastran\2023.1

install_dir	The full path to the directory used in the installation	
	Example:	The system RC file is *install_dir*/conf/nast20231rc.
doc_install_dir	The full path to the directory used in documentation/example installation.	
	Example:	`The documentation subdirectory is doc_install_dir/doc`
prod_ver	The Product and Version of MSC Nastran	
	Example:	`For MSC Nastran 2023.1 prod_ver=msc20231`
vernum	The version number. For MSC Nastran 2023.1, this is `20231`	
arch	The architecture of the platform.	

	Example:	win64i8 for Windows 64 version
	Example:	linux64i8 for Linux 64 version
nast_ver	The command used to run Nastran	
	Example:	`For MSC Nastran 2023.1 nast_ver=nast20231`

Document Structure

The IOG focuses on three areas of MSC Nastran use and also features additional information in the form of appendixes.

Note:	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 MSC Nastran users.

Installation and Configuration

Chapter 2 discusses the installation of MSC Nastran, while Chapter 3 demonstrates how to configure your system and MSC Nastran.

Basic and Advanced Use

Chapter 4 presents the basic functions of the nastran command and provides some details on how to use system files and databases. Chapter 5 explains how to use the advanced features of the nastran command and includes information on computer resource management.

Supplementary Information

In addition to these five chapters, the IOG also includes three appendixes. Appendix A contains the information to configure the runtime environment. Appendix B contains keywords and environmental variables and Appendix C contains system descriptions.

Supported Hardware and Operating Systems

Certified and Supported Operating Systems can be found at:

https://www.mscsoftware.com/platform-support

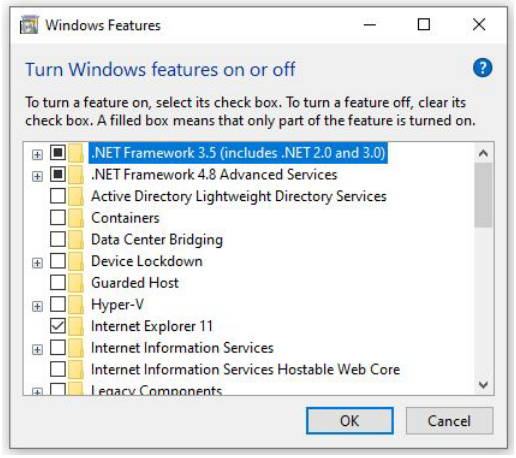
Vendor	FORTRAN Version	C Version	Default MPI
Linux (64-bit)	Intel oneAPI 2022.1.0	Intel oneAPI 2022.1.0	Intel MPI 2021.7.1
Microsoft (64-bit)	Intel oneAPI 2022.1.0	MSVC 14.31 Visual Studio 2022 17.1.7 Microsoft .NET 6	MircroSoft MPI 10

- This release uses the Cuda 11 toolkit to support the newest GPU accelerators from Nvidia, such as the A-100 (based on the Ampere microarchitecture). Nvidia drivers supporting Cuda 11 are required for GPU-accelerated analyses (450.36.06 or newer for Linux x86_64, 456.38 or newer for Windows).

- A fully supported OS has passed Hexagon's extensive QC process.

  Caveats:

  - Minimal testing has been made on a few unsupported OS's, but these OS's are not officially supported.
  - When running on Windows 10
    - Update the Windows 10 before installing the .Net framework.
    - Install .Net framework (might not be needed).
  - Enable .net framework (might not be needed).



- URL to download the .net framework.

  https://www.microsoft.com/en-us/download/confirmation.aspx?id=21

Note: For the latest information on supported platforms for upcoming releases of MSC products, please visit the following web site: https://hexagon.com/support-success/manufacturing-intelligence/design-engineering-support/platform-support

MSC Nastran Documentation Requirements

To view and navigate through the PDF based MSC Nastran Documentation, the following browsers are recommended.

Vendor	Desktop Environment	Browser	Browser Version
Linux (64-bit)	KDE	Adobe Reader	10.1.4 or higher
Linux (64-bit)	Gnome	Adobe Reader	10.1.4 or higher
Microsoft (64-bit)	Windows 10	Adobe Reader	10.1.4 or higher

Note: The browsers in the above table have been tested and work with the current version of the MSC Nastran Documentation.

Changes to MSC Nastran 2023

- Intel compilers have been updated and are oneAPI compilers.
- Certified platforms listed on the support page have had a full test suite of problems run. Supported platforms have had a minimum set of tests run on them.
- Supported/Certified platforms are now: RH 79, 86, SuSE12SP5, SuSE15SP3, Win10 (21H2, 22H2), Win11 (22H2).

The Directory Structure

The installation directory structure provides the following capabilities:

- Multiple versions of MSC products, such as the current and prior versions of MSC Nastran.
- Multiple computer architectures, such as linux64i8, win64i8, etc.

Note: If you create an installation by copying an existing installation, and the path is different than the installation you are copying from, then "rcmd" will need to be changed in the *install_dir*`/prod_ver/arch/nastran.ini` file to point to the new path.

Figure 1-1 shows the structure of the *install_dir* directory, which is selected during installation.

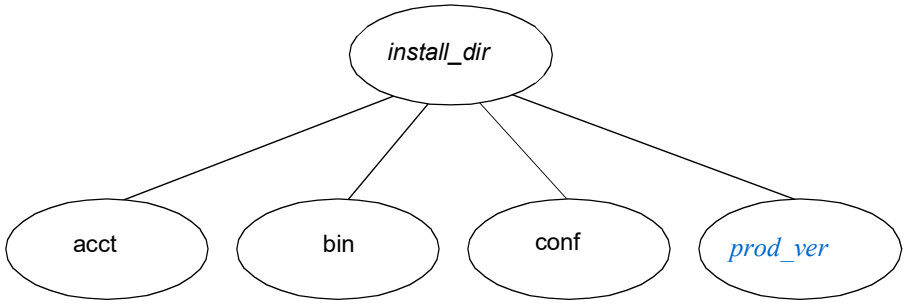


Figure 1-1 Directory for *install_dir*

Multiple Products Support

The MSC Nastran installation directory structure supports multiple products by using product-dependent and architecture dependent and independent directories and files. For example, Figure 1-2 shows that the *install_dir*/*prod_ver*/nast, actran and dytran directories on LINUX and *install_dir**prod_ver*\nast, actran and dytran directories on Windows contain the product-dependent files for MSC Nastran, Actran and Dytran respectively. All architecture-dependent files are located within the *arch* directory while the util and access directories contain the architecture-independent files for the various utilities and MSC.ACCESS. The utility programs directory (*install_dir*/*prod_ver*/util on LINUX and *install_dir**prod_ver*\util on Windows) contains source and make files for the utilities that are also delivered in source form. None of these files is architecture dependent.

Multiple Computer Architecture Support

The MSC Nastran installation directory structure also supports multiple computer architectures by using architecture-dependent directories and files. Several directories that hold architecture-dependent files are:

1. *install_dir*/*prod_ver*/arch on LINUX and *install_dir**prod_ver*\arch on Windows
2. *install_dir*/*prod_ver*/nast/beamlib/lib/arch, *install_dir*/*prod_ver*/nast/dr3/lib/arch, *install_dir*/*prod_ver*/nast/spline_server/lib/arch on LINUX and *install_dir**prod_ver*\nast\beamlib\lib\arch, *install_dir**prod_ver*\nast\dr3\lib\arch, *install_dir**prod_ver*\nast\spline_server\lib\arch on Windows.

where *arch* is the name of the architecture, e.g., linux64, Win64 (see Table 3-1).

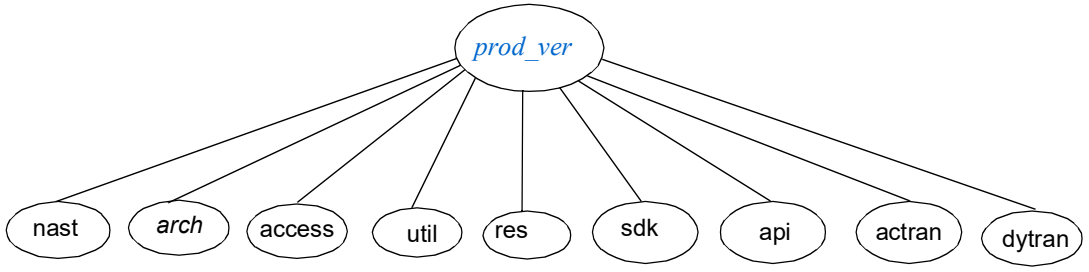


Figure 1-2 Directory for *prod_ver*

Building beam servers and dresp3 servers uses SCONS. Two new subdirectories are created in the *install_dir*/*prod_ver*/nast/ directory for building beam server, dresp3 server as shown in Figure 1-3. In addition, another new subdirectory is added for building spine server.

The subdirectory for beam library server: *install_dir*/*prod_ver*/nast/beamlib

The subdirectory for DRESP3 server: *install_dir*/*prod_ver*/nast/dr3

The subdirectory for Spline server: *install_dir*/*prod_ver*/nast/spline_server

The data structures of three external servers are described below. The ~lib/ directory for each server directory is architecture dependent.

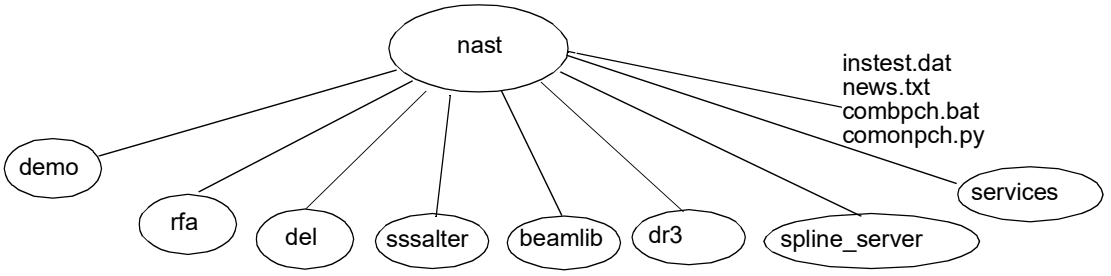


Figure 1-3 Directory for nast

The demo directory contains the demonstration problems.

The rfa directory contains rigid format alters.

The del directory contains DMAP for the delivery database

The beam server directory (*install_dir*/*prod_ver*/nast/beamlib on LINUX and *install_dir**prod_ver*\nast\beamlib on Windows) contains three SCons configuration files, include, library and source directories for the beam server sample programs. All architecture-dependent files are located within the lib directory.

The dr3 server directory (*install_dir*/*prod_ver*/nast/dr3 on LINUX and *install_dir**prod_ver*\nast\dr3 on Windows) contains three SCons configuration files, include, library and source directories for the dr3 server sample programs. None of theses files is architecture dependent except the lib directory.

The spline server directory (*install_dir*/*prod_ver*/nast/spline_server on LINUX and *install_dir**prod_ver*\nast\spline_server on Windows) contains three SCons configuration files, include, library and source directories for the spline server sample programs. None of theses files is architecture dependent except the lib directory.

The services directory contains examples of pre-defined User Defined Services (UDS).

Default Documentation Directories:

The documentation and examples are available on a separate installer that can be downloaded from SDC (MSC.Software Solutions Download Center) at https://mscsoftware.subscribenet.com. The default installation directories for documentation/ examples (*doc_install_dir*) are:

Windows:	C:\Program Files\MSC.Software\MSC_Nastran_Documentation\2023.1
Linux:	/msc/MSC_Nastran_Documentation/2023.1

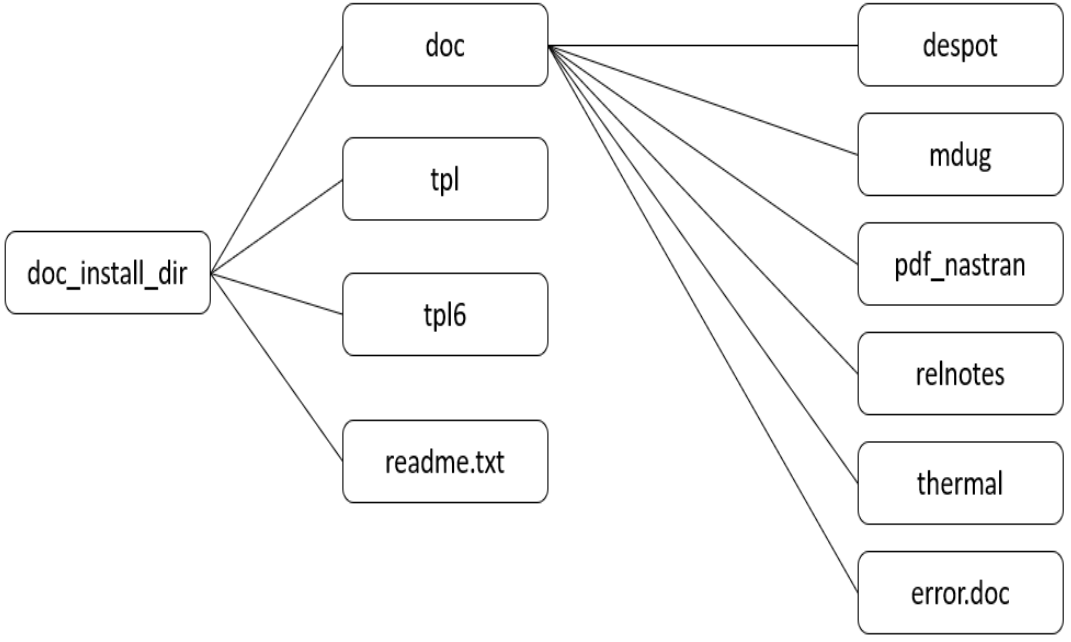


Figure 1-4 Directory for Documentation/ Example files

2 Installing MSC Nastran

- Overview
- Installing MSC Nastran on Linux Systems
- Installing MSC Nastran on Windows Systems

Overview

This chapter discusses the MSC Nastran interactive installation script, and includes installation procedures for LINUX and Windows systems.

Installing MSC Nastran on Linux Systems

This section begins with a brief set of installation notes and general information regarding MSC Nastran and MSC Licensing Helium or later. This section concludes with instructions on how to repeat a LINUX installation; this is useful when MSC Nastran is being installed on a number of computers.

GUI based (also known as standard or default), console and silent modes of running installer are currently supported. The GUI based mode requires a X windows environment and appropriately configured DISPLAY variable.

Installation Notes

- If you need a license file (served by FLEXlm), please contact your MSC account manager or MSC account administrator for assistance.

MSC Nastran

- Any run time libraries needed by MSC Nastran are included in this distribution.
- The installation test option will only be performed on the current architecture.
- If you install MSC Nastran in an installation base directory containing previous versions of MSC Nastran, your current settings for the “authorize”, “sdirectory”, “buffsize”, and “memory” keywords will be used as defaults.
- The installation directory should not contain non-standard characters such as “^” or “(“.
- To install MSC Nastran for Distributed Memory Parallel (DMP) operations, you must select one of the following three installation schemes if you want to use more than one host in a single MSC Nastran job:
  - Install MSC Nastran on a filesystem that is global to every host. This provides the easiest installation and system administration, but may present network load issues when the MSC Nastran is started and the delivery databases are being read.
  - Install MSC Nastran on every host on host-private file systems. This is harder to install and administer, but reduces the network load when MSC Nastran is started.
  - A combination of the above.

Note:	In all cases, the nastran command must have the same pathname, or be in the default PATH of every host that will run a DMP job. Recall that your “.profile” and “.login” files are not used for scp/rcp and ssh/rsh operations. MSC Nastran on windows needs to be installed on a network drive with no spaces to use multiple hosts.

FLEXlm License Server Version Helium of later

- In general, you should only install the FLEXlm License Server on one computer. Advanced licensing requirements may dictate more than one FLEXlm License Server.
- See Managing MSC Nastran Licensing for the systems supported by FLEXlm.
- If you have a FLEXlm network or counted node-lock license file, identify the name of the FLEXlm license server using "FLEXlm Server" option in the "Authorization Information" menu.
- If you have a node-lock authorization code file, identify the pathname and license file using the "Authorization File" option in the "Authorization Information" menu; the file will be appended to *install_dir*/conf/authorize.dat.

Note:	On Windows/Linux the licensing guide is available in the MSC Licensing\Helium folder as pdf file: 'licenseserver_usage_guide.pdf'.

Installation Procedure (GUI Based Installation)

The installer is self extracting binary that needs to be downloaded and run on your system to install all the necessary components of MSC Nastran. You can download the binaries from:

`https://mscsoftware.subscribenet.com`

Installing using Downloaded Files

1. Login as root
2. "cd" to a temporary directory with enough disk space. Create a subdirectory and "cd" into the subdirectory.
3. Download the delivery file from Solution Download Center. If you previously downloaded the file please proceed to the next step.
4. Executing the installation binary may require adding execution privilege:

   For MSC Nastran - `chmod +x nastran_20xx_<platform>.bin`
5. Execute the nastran_20xx_<platform>.bin script and follow the on screen prompts as described in the following steps.

   For Example, to execute the installation binary:

   ```
   ./nastran_20xx_linux64.bin
   ```

6. During the installation you will be prompted for several default options. These options are:

   Installation Directory- Where MSC Nastran will be installed

   Example: `/opt/msc`

   Default: /msc/MSC_Nastran/2023.1

   License Server - The server for MSC Nastran licensing

Example:`27500@node1`

Memory - The amount of memory MSC Nastran will dynamically allocate by default.

Example: `4gb`

Default: `max`

Buffsize - The size (in words) of MSC Nastran's internal I/O buffer.

Example: `16385`

Default: `32769`

Scratch Directory - Directory to use for temporary files This directory should be a large and local file system.

Example: `/scratch`

7. Cleanup: After installation is complete – you may remove the subdirectory created in Step 2 above.

Console Installation

The MSC Nastran installation supports console installations, which run in a xterm window with no graphical interface. Installations running in Console mode require the same input as the GUI based installer, but without needing the graphics display. MSC Nastran Console installations are generally used to facilitate installations on machines without graphics displays on your network

Installing using Downloaded Files

1. Login as root.
2. “cd” to a temporary directory with enough disk space. Create a subdirectory and “cd” into the subdirectory.
3. Download the delivery file from Solution Download Center. If you previously downloaded the file please proceed to the next step.
4. Executing the installation binary may require adding execution privilege:

   For MSC Nastran - `chmod +x nastran_20xx_<platform>.bin`
5. Execute the mscnastran_20xx_<platform>.bin script in console mode. Follow the on screen prompts (A detailed explanation of the Nastran paramters can be found throughout this document) for the remainder of the installation process.

   For Example, to execute the installation binary:

   ```
   ./nastran_20xx_linux64.bin --mode console
   ```

6. Cleanup: After installation is complete – you may remove the subdirectory created in Step 2 above.

Silent Installation

The MSC Nastran installation supports silent installations, which run in the background with no graphical interface or interaction with the desktop. Installations running in Silent mode rely on a pre-configured answer file `config.rec` to do the installation. Silent installations are generally used in a batch manner to facilitate installation on many machines on a network

Creating the Answer file

To create the answer file you need to run the MSC Nastran installation in normal (GUI) mode with a special switch which instructs the installation to record all of your answers in a specified answer file. The following example is for Linux. For other platforms use appropriate setup instead of mscnastran_20xx_<platform>.bin

To build a response file run the installer with the following options:

```
./nastran_20xx_linux64.bin --record
```

An on-screen popup message will show the directory where the answer file will be saved. The *config.rec* response file will be generated at the very end of the installer run.

Running the Silent mode installation

For the silent installation to run, the installer and the *config.rec* file must be in the same directory. The installer will automatically look for *config.rec* and start the installation. To run installation in silent mode use the following command:

```
./nastran_20xx_linux64.bin --mode silent
```

Installing MSC Nastran on Windows Systems

This section discusses the MSC Nastran Windows installation. The installation notes contain information regarding performance and disk space requirements, directory structures and setup information.

Note:	To install MSC Nastran on Windows, it is a required that you be a member of the Administrator group.

Installation Notes

- You must have one of the following systems to install and run MSC Nastran:
  - Windows 10-64, with at least 4 Gigabyte RAM, and 20 Gigabytes available disk space to install MSC Nastran.
  - Microsoft 2010 Redistributables are required. They may be downloaded from:

`http://www.microsoft.com/en-us/download/details.aspx?id=26999`

Note:	The disk space listed above is for installation of MSC Nastran only. Additional disk space is required to run MSC Nastran, dependent on the problem run.

- The following Microsoft Redistributables are installed if they do not already exist:
  - MSC_Install_VC_2010_X64(); - Microsoft Visual C++ 2010 x86 Redistributable - 10.0.30319
  - MSC_Install_VC_2012_X64(); - Microsoft Visual C++ 2012 Redistributable (x64) - 11.0.61030
  - MSC_Install_VC_2013_X64(); - Microsoft Visual C++ 2013 Redistributable (x64) - 12.0.21005
  - MSC_Install_VC_2015_X64(); - Microsoft Visual C++ 2015 Redistributable (x64) - 14.0.23026
- To build the Utility Programs using the supplied source, you must also have a suitable set of compilers. Refer to System Descriptions for details.
- The default directory (called the *install_dir*) for MSC Nastran products is "C:\Program Files\MSC Software\MSC_Nastran\2023.1". This can be changed to a new or existing directory of your choice.
- The default for the MSC Nastran scratch file directory is "c:\scratch". Having this directory on a separate drive from the system swap file can help performance.
- The default program group (folder) is named MSC.Software; you can have the icons installed in a different group if you choose. This group is created as a common group if the user doing the installation has administrator authority. Otherwise, this group is created as a private group.
- To run MSC Nastran from any directory, you must add the path *install_dir*\bin to your PATH. You can change your path in Windows by selecting the "control panel", and then "system". Then, click on the "Path" variable and add the following to text in the "Value" box.

  ```
  install_dir\bin
  ```

  Select "set", then "OK", and your path will be updated.

Installation Procedure

If you are downloading from the Solutions Download Center, download the self-extracting archive (.exe). You can download the binaries from:

`https://mscsoftware.subscribenet.com`

Then follow these steps:

1. Download the self-extracting installer (.exe) file to a subdirectory with enough disk space where the file can be executed.
2. Double clicking on the Product Installer will start the installation process.
3. During the installation you will be prompted for several default options. These options are:

   Installation Directory- Where MSC Nastran will be installed

   Example: `C:\msc`

   Default: `C:\Program Files\MSC.Software\MSC_Nastran\2023.1`

   License Server - The server for MSC Nastran licensing

Example:`27500@node1`

Memory - The amount of memory MSC Nastran will dynamically allocateby default.

Example: `4gb`

Default: `max`

Buffsize - The size (in words) of MSC Nastran's internal I/O buffer.

Example: `16385`

Default: `32769`

Scratch Directory - Directory to use for temporary files This directory should be a large and local file system.

Example: `C:\scratch`

4. Cleanup: You may remove the installer file from the subdirectory used in step 1 after the installation is complete. If you remove the installer, you will have to download or copy the installer back onto your computer to repair or modify your MSC Nastran installation. Uninstalling MSC Nastran can be done using either the installer, or from Add/Remove Programs in the Control Panel.

Silent Installation

The MSC Nastran installation supports silent installations, which run in the background with no graphical interface or interaction with the desktop. Installations running in Silent mode rely on a pre-configured answer file `file.iss` to do the installation. Silent installations are generally used in a batch manner to facilitate installation on many machines on a network.

Creating the Answer file

To create the answer file you need to run the MSC Nastran installation in normal (GUI) mode with a special switch which instructs the installation to record all of your answers in a specified answer file. The following example is for windows.

To build a response file run the installer with the following options:

```
./nastran_20xx_win64.exe /r /f1"Full_path_to_file.iss"
```

The `FILE.iss` response file will be generated at the very end of the installer run.

Running the Silent Mode Installation

For the silent installation to run, the installer and the FILE.iss file must be in the same directory. To run installation in silent mode use the following command:

```
./nastran_20xx_win64.bin /s /f1"Full_path_to_file.iss"
```

3 Configuring MSC Nastran

- Overview
- System-Specific Tuning
- Using the “msc20231” Command
- Using the “mscinfo” Command (LINUX)
- Managing MSC Nastran Licensing
- Activating MSC Nastran Accounting
- Determining System Limits
- Managing Remote and Distributed Hosts
- Limiting “memory” Requests
- Customizing the News File
- Customizing the Message Catalog
- Defining a Computer Model Name and CONFIG Number
- Generating a Timing Block for a New Computer
- Customizing Queue Commands (LINUX)
- Customizing the Templates
- Using Regular Expressions

Overview

This chapter is intended for system administrators or anyone who needs to manage an MSC Nastran installation. It starts with information on tuning your system for better performance. The chapter then concentrates on configuring MSC Nastran for your system. Licensing must be configured before MSC Nastran will run. Other items that may require configuration include system resource limits, the command initialization file, runtime configuration files, timing blocks, and queue commands.

Two documentation conventions are used throughout the remainder of this document (typically in directory specifications). The string "*install_dir*" indicates the directory where MSC Nastran was installed. On LINUX, this might be "/msc/MSC_Nastran/2023.1" and on Windows "C:\Program Files\MSC.Software\MSC_Nastran\2023.1". The string "*arch*" indicates the Hexagon architecture name for your computer; they are generally based on the operating system name on LINUX, while on Windows, they describe the processor. The architectures are as follows:

Table 3-1 Architecture Names

Computer	arch
Intel Linux x86-64	linux64i8
Intel Windows x86-64	win64i8

Throughout this document, while file pathnames and sample commands for Windows systems will use the standard backslash "\" directory separator character, MSC Nastran also accepts pathnames using the slash "/" character as a replacement.

Note:	On Windows operating systems, the command shell, CMD.EXE does not accept slash "/" characters as the first character in a pathname.

System-Specific Tuning

This section presents some information on system-specific tuning that can help MSC Nastran performance.

All Systems

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

MSC 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 MSC Nastran performance than a faster processor with a slower memory bus.

Linux

- Linux uses memory to cache I/O automatically. Increasing memory may reduce I/O time.
- Do not use a network drive for scratch. Using network drives may cause performance issues

Windows 10

Hexagon has found performance issues on Windows with models greater than 100 thousand DOF. These issues may be addressed using MAPIO options. Please see Using File Mapping in the *MSC Nastran Installation and Operations Guide* for more information.

Systems Running on Intel® Processors with Hyper Threading

The Intel® Xeon® processor introduces a feature called Hyper Threading, where a single physical processor can support more than one logical instruction stream, simulating multiple logical processors on a single physical processor. For many applications and environments, this capability may offer performance improvements over non-Hyper Threading processors. If multiple MSC Nastran analysis jobs are running concurrently, however, there may be performance degradations. If an installation determines this to be the case, hyper threading should be disabled.

Using the "msc20231" Command

The "msc20231" command is shown as a prefix for most of the programs and commands described in this document, for example:

```
msc20231 nastran ...
```

Note: For simplicity, the symbol *prod_ver* will be used for msc20231.

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

The `msc20231` command is located in

install_dir`/bin/`*prod_ver*

on LINUX, and

install_dir`\bin\`*prod_ver*

on Windows.

Using the “mscinfo” Command (LINUX)

The “mscinfo” command is available on LINUX systems to display various hardware and software configuration info. This utility is run with the command

```
prod_ver mscinfo
```

mscinfo will display hardware and software configuration report, including

- Hostname.
- MSCID.
- 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, e.g. the partition size of /tmp directory.
- Local disk sizes.

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

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

Managing MSC Nastran Licensing

Note:	On Windows/Linux the licensing guide is available in the MSC Licensing\Helium folder as pdf file: ‘licenseserver_usage_guide.pdf’. The default FLEXlm folder is C:\Program Files\MSC.Software\MSC Licensing\Helium on Windows and /msc/MSC.Software/MSC Licensing/Helium on Linux

In order to run, MSC Nastran requires one of the following licensing methods:

- The name of a network license server (if your computer supports FLEXlm).
- The pathname of a file containing FLEXlm licenses (if your computer supports FLEXlm).
- The pathname of a file containing one or more node-locked authorization codes.

When selecting the licensing method, MSC Nastran will use the first non-null value that it finds in the following hierarchy:

1. The value of the authorize keyword on the command line.
2. The value of the MSC_LICENSE_FILE environment variable.
3. The value of the "authorize" keyword in an RC file.

If a non-null value cannot be found, the following User Fatal Message (UFM) is displayed by the nastran command:

```
*** USER FATAL MESSAGE (nastran.validate_authorize)
    authorize=""     (program default)

    The keyword shall not be blank or null.
```

UFM 3060

If a non-null value is found for the "authorize" keyword, your MSC Nastran job will be started. If the licensing information is later determined to be invalid or insufficient for the analysis, a UFM 3060 error message is printed in the .f06 file:

```
*** USER FATAL MESSAGE 3060, SUBROUTINE MODEL - OPTION opt NOT IN APPROVED LIST.
     SYSTEM DATE (MM/DD/YY): mm/dd/yy
     SYSTEM MSCID: 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. Other information pertinent to this failure will be found in the LOG file.

FLEXlm Licensing

FLEXlm is available on the following MSC Nastran platforms:

- Intel - Linux x86-64
- Intel - Windows

Clients with network-licensed Hexagon installations are encouraged to employ the most recent versions of the FLEXlm and MSC licensing daemons (lmgrd/lmutil/msc).

The binaries maintain downward compatibility, and regular upgrades are recommended, regardless of whether the current software application level required the upgrade. Updates are available at:

FLEXlm utilities are available at:

Windows:

The msc_licensing_lithium_windows64.exe is available at:

https://mscsoftware.subscribenet.com/control/mnsc/product?child_plneID=593623

Linux:

The msc_licensing_lithium_linux64.bin at:

https://mscsoftware.subscribenet.com/control/mnsc/download?element=7087407

A license server on either LINUX or Windows can serve licenses for any number of LINUX and/or Windows systems.

Additional information about configuring and running FLEXlm with MSC Nastran is available in the MSC Licensing Lithium Guide. This guide is available at:

https://mscsoftware.subscribenet.com/control/mnsc/download?element=7087407

A fact sheet is also available to help users and system administrators to configure and manage the FLEXlm licensing package.

https://simcompanion.mscsoftware.com/infocenter/index?page=content&id=DOC1095

Using FLEXlm Licensing

The following table describes various keywords that control MSC Nastran's licensing subsystem.

Table 3-2 MSC Nastran Keywords Related to Licensing

Keyword	Comments
authorize	The license specification.
authque	The number of minutes to wait for licenses if the license server cannot immediately honor a license request, with 0 (zero) indicating no licensing queuing. If not specified, 20 minutes is the MSC Nastran default.
authinfo	The level of licensing diagnostic messages written to the MSC Nastran log file, in the range of 0-9 with 0 indicating minimal diagnostics, and 9 indicating extensive diagnostic output.
a.port	The default port number for FLEXlm licensing. The default value is "27500". If a.port is set to an integer value greater than 0, FLEXlm license specifications in the form "@node" are converted to "port@node", where port is the value of the keyword a.port. If a.port is set to the value "no" or "0", FLEXlm license specifications in the form "@node" are passed to the licensing subsystem without change. This allows use of the FLEXlm "default port scanning" feature.

The "authorize" keyword is used to indicate the licensing source. The value can be any of the following:

Value	Comments
@node	The specified node is the license server using the default port number 27500. See the description of the a.port keyword above for details.
port@node	The specified node is running a license server listening on the specified port.
filename	The specified file is used for authorization. This file may contain FLEXlm licensing information for either a node-locked or network license.
value:value:value on Linux *or* *value;value;value* on windows	A quorum of three redundant FLEXlm license server nodes.
value:*value*:...	LINUX: A list of FLEXlm licensing files, license server nodes, or quorums.
value;*value*;...	Windows: A list of FLEXlm licensing files, license server nodes, or quorums.

Examples are:

```
auth=/msc/MSC.Software/MSC Licensing/Helium/license.dat
```

on a LINUX system, and

```
auth=c:\Program Files\MSC.Software\MSC Licensing\Helium\license.dat
```

on a Windows system, the specified FLEXlm license file will be used. If this license file contains one or more “SERVER” lines, the file is only used to identify the server(s). If not, the file will be treated as a FLEXlm node-lock license file.

```
auth=@troll
```

Node “troll” is a FLEXlm license server using the default port number. If a.port is set to "no", node "troll" is a FLEXlm license server using a port number in the FLEXlm default range of 27000-27009.

```
auth=27500@troll
```

Node “troll” is a FLEXlm license server using the specified port number.

For LINUX:

```
auth=27500@banana1:27500@banana2
```

For Windows:

```
auth=27500@banana1;27500@banana2
```

Two alternate network license servers, “banana1" and “banana2", will be used to provide network licensing services.

Activating MSC Nastran Accounting

MSC 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 on LINUX systems and *install_dir*\acct on Windows systems.

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

To activate MSC Nastran accounting, set the keyword “acct=yes” in any RC file or on the command line. Placing the keyword in the System RC file will enable accounting for all jobs. The location of the System RC files is described in Command Initialization and Runtime Configuration Files, 138 in Appendix A.

Instructions for generating usage summaries from the MSC accounting data are provided in the section titled Using the Basic Keywords, 62.

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” (page 172) in the command initiation file (nastran.ini) or a system RC file.
2. The account validation keyword, “acvalid” (page 172), can be used to validate the “acid” keyword. If “acvalid” is not defined in the command initialization file, MSC Nastran will not require the “acid” keyword; if the “acvalid” keyword is defined, MSC Nastran will require a valid “acid”. See Enabling Account ID Validation, 38 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. Specifying Parameters, 138 in Appendix A 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, 55). 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.

```
nast_ver example
```

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

```
nast_ver 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.

```
nast_ver example
```

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

```
nast_ver 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

```
nast_ver example
```

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

```
nast_ver example acid=123
```

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

```
nast_ver 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. The Korn shell version is primarily intended for LINUX systems; the Perl version can be used on any LINUX or Windows systems that have Perl installed.

Note:	You must have Perl installed on your system to use the Perl sample account validation program. Perl is available from numerous sources, including the URL
	http://www.perl.com
	This is not an Hexagon MI site and Hexagon has no control over the site's content. Hexagon cannot guarantee the accuracy of the information on this site and will not be liable for any misleading or incorrect information obtained from this site.

The Korn shell version is:

```
#!/bin/ksh
#
# THIS PROGRAM IS CONFIDENTIAL AND A TRADE SECRET OF Hexagon AB AND/OR ITS
# SUBSIDIARIES.  THE RECEIPT OR POSSESSION OF THIS PROGRAM DOES
# NOT CONVEY ANY RIGHTS TO REPRODUCE OR DISCLOSE ITS CONTENTS,
# SELL, LEASE, OR OTHERWISE TRANSFER IT TO ANY THIRD PARTY,
# IN WHOLE OR IN PART, WITHOUT THE SPECIFIC WRITTEN CONSENT OF
# Hexagon AB AND/OR ITS SUBSIDIARIES.
#
# 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
```

On LINUX, this program is activated with the following

```
acvalid=`install_dir/bin/checkac install_dir/acct/account.dat %acid%`
```

The Perl version is:

```
#!/usr/local/bin/perl
#
# THIS PROGRAM IS CONFIDENTIAL AND A TRADE SECRET OF Hexagon AB AND/OR ITS
# SUBSIDIARIES.  THE RECEIPT OR POSSESSION OF THIS PROGRAM DOES
# NOT CONVEY ANY RIGHTS TO REPRODUCE OR DISCLOSE ITS CONTENTS,
# SELL, LEASE, OR OTHERWISE TRANSFER IT TO ANY THIRD PARTY,
# IN WHOLE OR IN PART, WITHOUT THE SPECIFIC WRITTEN CONSENT OF
# Hexagon AB AND/OR ITS SUBSIDIARIES.
#
# 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 = <AC> ) {
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
```

LINUX

LINUX sites can also 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*/*prod_ver*/*arch*/acct, a “set uid” program.

Note: Before making *install_dir*/*prod_ver*/*arch*/acct a set-uid program, Hexagon recommends that you carefully review the *install_dir*/*prod_ver*/util/mscact.c source code, ensure that you have built *install_dir*/*prod_ver*/*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/prod_ver/arch/acct
chgrp secure-group install_dir/prod_ver/arch/acct
chmod ug+s install_dir/prod_ver/*/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.

Determining System Limits

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

```
nast_ver limits
```

On LINUX, the resource limits on a remote computer that has MSC Nastran installed are obtained with:

```
nast_ver limits node=remote_computer
```

Note:	1. The limits can vary among users and computers. If a queuing system such as LSF or PBS is installed, different limits may also be found on the various queues.
	2. The output from the limits special function may specify “unlimited” on LINUX systems. 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.

Sample output from this command for the various computers used to port MSC Nastran follows.

Windows

Current resource limits:		
Physical memory:	8125	MB
Physical memory available:	4165	MB
Paging file size:	16249	MB
Paging file size available:	12306	MB
Virtual memory:	8388	MB
Virtual memory available:	8388	MB

Intel 64 - Linux

Current resource limits:	
CPU time:	unlimited
Virtual address space:	unlimited
Working set size:	unlimited
Data segment size:	unlimited
Stack size:	10240 KB
Number of open files:	1024 (hard limit:1024)
File size:	unlimited
Core dump filesize:	0 MB

Managing Remote and Distributed Hosts

Your site can control the hosts available to remote and distributed (DMP) jobs by creating host “accept” or “deny” utilities that list the hosts that a remote or DMP job may or may not use respectively.

A site administrator may create utilities for remote and DMP acceptance or denial nodes. These utilities could be placed in *install_dir*/*prod_ver*/*arch*/ and called: `rmtaccept`, `rmtdeny`, `dmpaccept`, `dmpdeny`. They would need to return the list of nodes for acceptance or denial for remote or DMP submittals.

The “rmtdeny” and “dmpdeny” utilities list those hosts that cannot be used by a remote or DMP job. The “rmtaccept” and “dmpaccept” utilities lists those hosts that can be used by a remote or DMP job. At most one and only one of these utilities will be used. The nastran command will first look for the “deny” utility. If it exists and is executable, it will be run and its stdout parsed — any host listed cannot be selected by the job. If the “deny” utility does not exist, the nastran command will look for the “accept” utility. If it exists and is executable, it will be run and its stdout parsed — only those hosts listed can be selected by the job. If neither utility exists, any host will be accepted.

The required output format of these utilities is one host per line of output. For example, consider the following output:

```
banana1
banana2
```

If written by a “deny” utility, neither “banana1” nor “banana2” will be available to an MSC Nastran job; if written by an “accept” utility, only these two hosts will be available.

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 value is:

```
memorymaximum=0.5*physical
```

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

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.

Customizing the News File

MSC delivers a news file (*install_dir*/*prod_ver*/nast/news.txt on LINUX and *install_dir**prod_ver*\nast\news.txt on Windows) that briefly describes important new features of the release. You can also use news file to distribute information to the users of MSC Nastran.

There are two ways the news file can be viewed. The most common way is by specifying "news=yes" or "news=no" 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

```
nast_ver news
```

This will display the news file on the screen.

Customizing the Message Catalog

MSC Nastran uses a message catalog for many messages displayed in the .f06 file. The standard message catalog source file is

```
install_dir/prod_ver/util/analysis.txt
```

on LINUX and

```
install_dir\prod_ver\util\analysis.txt
```

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

When reviewing the analysis.txt file, note the use of system(319), also called keyword XMSG, to provide a mechanism for printing additional information associated with messages. If the "I" in Information is a lower case "i" and XMSG=1, then the additional information will be printed.

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

```
prod_ver 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 (p. 193), and can be tested using the command

```
nast_ver msgcat=myfile.msg other_nastran_keywords
```

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

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

on LINUX, or

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

on Windows, 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.

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 MSC 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 (page 176) in the node RC file; see Command Initialization and Runtime Configuration Files, 138 in Appendix A and Customizing Command Initialization and Runtime Configuration Files, 151 in Appendix A. Note, however, this will not set a model name.

Generating a Timing Block for a New Computer

Generating timing blocks have been depreciated because the Multiply Add method selection has been changed.

Customizing Queue Commands (LINUX)

The nastran command runs an MSC Nastran job by validating the command line and RC files, generating a "job script" that will run the MSC 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 (p. 209), 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:	1. When defining queue commands, it may be useful to build the job script but not actually execute it. Use the "-n" option, for example *prod_ver* `-n nastran myjob queue=myqueue`
	2. The examples presented below are only intended to illustrate the "submit", "qsub" and "queue" keywords. The examples may not work with your queuing software.
	3. The Korn shell must be used 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
cputime	Value specified with the “cputime” 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, (%cputime% - %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

```
nast_ver 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 Examples, 53. 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

```
nast_ver 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

```
nast_ver 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:	
	1. If the first character of the command is the LINUX pipe character, “|”, the contents of job script will be piped into the command.
	2. 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.

Customizing the Templates

The nastran command relies on several templates to construct the job script (LINUX) or control file (Windows) that is built for every MSC Nastran job. Note that, for LINUX, the job script includes the necessary commands to build the control file. Several templates are provided:

For LINUX, the following files are used. Note that the installed template files are the same for all architectures. The file names in the *arch* directory are linked to files in the bin directory.

- *install_dir*/*prod_ver*/*arch*/nastran.dmp is used for DMP jobs.
  The keyword defining this file name is 0.dmp.
- *install_dir*/*prod_ver*/*arch*/nastran.lcl is used for serial or SMP jobs run on the local system.
  The keyword defining this file name is 0.lcl.
- *install_dir*/*prod_ver*/*arch*/nastran.rmt is used for serial or SMP jobs run on a remote system using the "node" keyword. The keyword defining this file name is 0.rmt.
- *install_dir*/*prod_ver*/*arch*/nastran.srv is used for Toolkit jobs.
  The keyword defining this file name is 0.srv.

The templates provided by Hexagon support all versions of MSC Nastran since MSC Nastran 68.0 for all LINUX platforms.

For Windows,

- *install_dir**prod_ver**arch*\nastran.lcl is used for serial or SMP jobs run on the local system.
  The keyword defining this file name is 0.lcl.
- *install_dir**prod_ver**arch*\nastran.rmt is used for serial or SMP jobs run on a remote system using the "node" keyword. Currently, the remote system *must* be a LINUX system running the "rshd" daemon.
  The keyword defining this file name is 0.rmt.
- *install_dir**prod_ver**arch*\nastran.srv is used for Toolkit jobs.
  The keyword defining this file name is 0.srv.

These templates may be modified to suit your needs. Please make backup copies if you modify these files. You may also use the appropriate keyword, specified in either the INI file or on the command line, to specify the location of your modified template file.

Note:	When customizing the templates, it may be useful to build the job script or control file but not actually execute it. Use the "-n" option, e.g., *nast_ver* `-n myjob` The name of the generated file will be echoed to stdout.

Keyword Reference Syntax

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

Table 3-3 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, 55.
%*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 the LINUX "*install_dir*/msc20231/MSC_ARCH/nastran.lcl" file. The same syntax is supported for the Windows control file templates.

Unconditional Keyword Substitution

```
export MSC_BASE="%MSC_BASE%"
```

The keyword reference "MSC_BASE" will be replaced by the value of the "MSC_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

```
%MSC_ARCH=linux:%startdate=date +%%a %%h %%d %%H:%%M:%%S %%Z %%Y
%MSC_ARCH!linux:%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 "MSC_ARCH" keyword contains the string "linux" while the remainder of the second line will be included if "MSC_ARCH" does not contain the string "linux". 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=2600 MHz:% echo Node 1 % s.model%,
...%s.model=2330 MHz:% echo Node 2 % s.model%,
...%s.model=%
```

if “s.model” is “2600 MHz” then the following is printed:

```
Node 1 2600 MHz
```

if “s.model” is “2330 MHz” then the following is printed:

```
Node 2 2330 MHz
```

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>2014.0: %CONFIG=%config%
```

The CONFIG statement will be included if “a.release” is greater than 2014.

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.

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. The syntax follows.

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 left most 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.

4 Using the Basic Functions of MSC Nastran

- Overview
- Using nast20231 Command
- Using the Basic Keywords
- Specifying Memory Sizes
- Determining Resource Requirements
- Using the Test Problem Libraries
- Making File Assignments
- Using Databases
- Using the INCLUDE Statement
- Using the SSS Alter Library
- Resolving Abnormal Terminations

Overview

This chapter is directed to the engineer running MSC Nastran, and discusses how the basic functions of MSC Nastran are used. 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.

Using nast20231 Command

MSC Nastran is executed from the command line using the command `nast20231`. This command sets environment variables in order for MSC Nastran to correctly execute on your system and it also has capabilities to predict memory and solvers (as of MSC Nastran 2023.1 with `solve=auto`). The basic format of the `nast20231` command is:

`nast20231` *input_data_file* `keywords`

`nast20231` *input_data_file* [keyword1=value1 keyword2=value2 …]

where input_data_file is the name of the file containing the input data and keyword=valuei is one or more optional keyword assignment arguments. For example, to run an MSC Nastran job using the data file example1.dat, enter the following command:

```
nast20231 example1.dat
```

Various options to the `nast20231` command are available using keywords. Keyword assignments consist of a keyword, following by an equal sign, followed by a keyword value. For example,

```
nast20231 example1.dat memorymax=16gb
```

Note:	In Windows you can use a hash mark # instead of the equal sign. This is useful if `nast20231 example1.dat memorymax=16gb` command is placed in `.bat file`. `nast20231 example1.dat memorymax=16gb`

The details of submitting an MSC Nastran job are specific to your machine. Contact your IT personnel or refer to the MSC Nastran Installation Guide for further information. Keyword assignments can be specified on the command line or included in RC files.

The following sets of RC files are controlled by you:

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

The locations and names of these RC files are described in Command Initialization and Runtime Configuration Files, 138 in Appendix A.

Note:	
	1. The LINUX tilde (~) shorthand is not recognized within RC files.
	2. Environment variables are only recognized when used in the context of a logical symbol (see Using Filenames and Logical Symbols, 60) or when used to initialize user defined keyword (see User-Defined Keywords, 143 in Appendix A).
	3. When a keyword is specified on the command line, embedded spaces or special characters that are significant to the shell must be enclosed in quote marks; quotes marks should not be used within RC files unless they are significant to the keyword's value.

File Types and Versioning

MSC Nastran's default input and output files use the following types: For a more comprehensive list, refer to FORTRAN Files and Their Default Attributes (p. 59) in the *MSC Nastran Quick Reference Guide*.

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

Using Filenames and Logical Symbols

Several of the parameters used by MSC Nastran, including command line arguments, initialization and RC file commands, and statements within MSC 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:	
	1. 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.
	2. 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.
	3. MSC Nastran will accept 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

```
nast_ver DATADIR:nastran example
```

Since MSC Nastran automatically sets the OUTDIR environment variable to the value of the "out" keyword, if DATADIR is defined as above, 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, 210 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:

```
nast_ver help
```

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

```
nast_ver help keyword1 [keyword2 ...]
```

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

```
nast_ver limits
```

This request will display the current LINUX resource limits.

```
nast_ver news
```

This request will display the news file.

```
nast_ver system
```

This request will display system information about the current computer.

On LINUX, these requests can be executed on a remote computer that has MSC Nastran installed by also specifying the keyword "node=*nodename*", for example:

```
nast_ver system node=thatnode
```

Using the Basic Keywords

The following table is 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, 86, and a complete list of all keywords and their syntax is listed in Keywords, 172.

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.

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

Specifying Memory Sizes

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

Table 4-1 Memory Size Specifications

Specification	Size (Words)
nT, nTW	$n(1024^4)$
nTB	$n\frac{(1024^4)}{bpw}$
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	$\frac{n(1024)}{bpw}$
n*physical, $n$xphysical	$n \cdot memory_{physical}$
n*virtual, $n$xvirtual	$n \cdot memory_{virtual}$

where: $bpw = 8$; "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.

Note: In order to use the "physical" and "virtual" specifications, the computer's physical memory and swap file size must be known to the nastran command. The nastran command always knows both these sizes. The computer's physical and virtual memory sizes can also be set via the "s.pmem" and "s.vmem" keywords respectively.

Examples are

```
nast_ver example memory=1gb
```

Set the memory request to one gigabyte, 1024 megabytes, 1048576 kilobytes, 1073741824 bytes, 134217728 words.

```
nast_ver example memory=0.5xPhys
```

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

Maximum Memory Size

Table 4-2 lists the maximum "memory" size for MSC 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 physical memory systems and 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, listed in System Descriptions (App. C), and space required for the various operating system and Fortran runtime libraries. Jobs submitted with mode=i8 on platforms that support it, have unlimited memory.

Table 4-2 Maximum Memory Size

Platform	Memory
Linux64	limited by system limits and virtual address space
Win64	limited by system limits and virtual address space

Determining Resource Requirements

For most models of moderate size (up to 500,000 grid points for static analysis), you need not be concerned with resource requirements since the default MSC 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.

There are several tools available to assist you in determining the resource requirements of your job, however using mem=max is the best way to reduce solver and I/O times. Table 4-3 and Table 4-4 are the simplest tools, they present gross estimates of the memory and total disk space requirements of static analyses using default parameters with normal output requests. Other solution sequences will generally have greater requirements.

Table 4-3 Estimated Memory Requirements of Static Analyses

Degrees of Freedom	Memory Requirements
	Others
$DOF \leq 100000$	10 Mb
$100000 < DOF \leq 400000$	50 Mb
$400000 < DOF \leq 1000000$	1 Gb

Table 4-4 Estimated Total Disk Requirements of Static Analyses

Degrees of Freedom	Total Disk Space Requirements
$DOF \leq 100000$	1 Gb
$100000 < DOF \leq 400000$	4 Gb
$400000 < DOF \leq 1000000$	10 Gb

More detailed resource estimates can be obtained from the ESTIMATE program. 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. The number of modes 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, described in User Information Messages 4157 and 6439, 100, but this requires an MSC Nastran run.

Estimating BUFFSIZE

Table 4-5 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 4-5 Suggested BUFFSIZE Values

Degrees of Freedom	BUFFSIZE
$DOF \leq 100000$	8193
$100000 < DOF \leq 400000$	16385
$DOF > 400000$	32769

Note:	The actual I/O transfer size is $(BUFFSIZE - 1) \times bpw$ where bpw is 8.

Using the Test Problem Libraries

One library of test problems are delivered with MSC Nastran. It is only installed with the "Complete" installation option.

- These files are accessible via the DEMODIR symbol, or via the path *install_dir*/*prod_ver*/nast/demo on LINUX and *install_dir**prod_ver*\nast\demo on Windows.
- The test problem library (TPL) contains a general selection of MSC Nastran input files showing examples of most of the MSC Nastran capabilities. This directory of test files is on a separate documentation/ example installation. In general, these files are not documented. The files are accessible via the TPLDIR symbol, or via the path *doc_install_dir*/tpl (or tpl6) on LINUX, and *doc_install_dir*\tpl (or tpl6) 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,

```
nast_ver 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, 75.

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.

Making File Assignments

Using the ASSIGN statement in your input file, you can assign physical files used by MSC Nastran to FORTRAN units or DBset files or you can modify the properties of existing or default file assignments. The ASSIGN statement is documented in the File Management Statements (Ch. 3) in the *MSC Nastran Quick Reference GuideMSC Nastran Quick Reference Guide*.

ASSIGN Statement for FORTRAN Files

For FORTRAN files, the format of the ASSIGN statement is

```
ASSIGN logical-key[={filename|*}] [UNIT=u] [[STATUS=]{NEW|OLD|UNKNOWN}]
[[FORM=]{FORMATTED|UNFORMATTED|BIGENDIAN|LITTLEENDIAN|LTLEND|<ostype>}] [DEFER]
[{TEMP|DELZERO}] [DELETE] [SYS='sys-spec']
```

Currently, there are no values of the SYS field defined for FORTRAN files on any system. For a list of the FORTRAN files and their default attributes, please refer to Table 3-1 in the File Management Statements (Ch. 3) in the *MSC Nastran Quick Reference Guide*. For more information about byte-ordering within binary files (the "endian" of a file), please refer to Binary File Byte Ordering (Endian) (App. C).

ASSIGN Statement for DBsets

ASSIGN *logical-name*[=*filename*|*}] [TEMP] [DELETE] [SYS='sys-spec']

See Using the SYS Field, 91 for details on the SYS field for DBsets.

Scratch DB Set Names

The default base name for scratch DB Sets 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.

LINUX:	*nast_ver* example sdir=/tmp
Windows:	*nast_ver* 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.

The following tables give information about the DBALL and SCRATCH DBset default allocations.

DBset	Memory			BUFFSIZE		Physical File Attribute		
	Type	Size	Units	Assignable	Size	Logical Name	Physical Name	Size
MASTER	RAM	200000	Words	YES	32769	MASTER	*dbs*.MASTER	5000
DBALL	N/A	-		YES	32769	DBALL	*dbs*.DBALL	See Table 4-6
OBJSCR	N/A	-		NO	32769	OBJSCR	*sdir*.OBJSCR	5000
SCRATCH	SMEM	See note	GINO Blocks	YES	32769	SCRATCH	*sdir*.SCRATCH	See Table 4-6
SCRATCH	N/A	-		YES	32769	SCR300	*sdir*.SCR300	See Table 4-6
User DBset	N/A	-		YES	32769	DBset	dbs.DBset	25000

Note: The default SMEM value is 100 GINO Block (1 BLOCK= 1 BUFFSIZE) for all platforms.

where:

DBset	The DBset name.
Memory	The size of memory (in words) used by 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.

Table 4-6

Default Maximum DBALL and SCRATCH DBset Sizes in GINO Blocks			
Memory (MEM)	BUFFSIZE		
	BUFFSIZE < 32769	32769 ≤ BUFFSIZE < 65537	BUFFSIZE = 65537
MEM < 512mb	500,000	1,000,000	1,000,000
512mb ≤ MEM < 4096mb	1,000,000	2,000,000	2,000,000
MEM ≥ 4096mb	10,000,000	20,000,000	20,000,000

Note: These values will be reduced, if necessary and without any information messages, to the maximum file size supported by the file system on which the file was allocated.

Default Maximum DBALL and SCRATCH DBset Sizes in GB for Specific BUFFSIZE Values			
Memory (MEM)	BUFFSIZE		
	8193	32769	65537
MEM < 512mb	30 Gb	244Gb	488Gb
512mb ≤ MEM < 4096mb	61 Gb	488Gb	976Gb
MEM ≥ 4096mb	610 Gb	4882Gb	9765Gb

Using Databases

MSC Nastran uses 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 MSC 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, 61.
- 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.

Table 4-7 Database I/O Capabilities

Computer	File Mapping	Buffered I/O	Async I/O
Intel Linux64	No	Yes	Yes
Intel Windows	Yes	Yes	Yes

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 automatically 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, 71.
2. Using ASSIGN statements in the FMS Section of the input data file. See ASSIGN Statement for DBsets, 68 and Using the ASSIGN Statement, 73.

Using the "dbs" Keyword

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

```
ID MSC, 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

```
nast_ver TPLDIR:am76/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:

```
nast_ver TPLDIR:am76/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 MSC, 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.
ENDDATA $ 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 MSC, 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:

```
nast_ver TPLDIR:am76/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 MSC,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:

```
nast_ver TPLDIR:am76/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 The File Management Section (FMS) (p. 47) in the *MSC 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 maximum size of 2000 GINO blocks for DBALL1 and a maximum size of 300 kilobytes for DBALL2. The maximum 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 8. 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, 63.

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 spaces, commas, special characters or dollar signs or, on LINUX; lowercase characters. for example,

```
INCLUDE 'filename'
```

The file name may include a directory specification, where directory levels are indicated using a using a directory level separator character ("/" on Linux and "/" or "\" on Windows).

Note: The RFINCLUDE and RFALTER statements may be used instead of the INCLUDE statement to insert a specified file into the input file. Except for the directory searching order specified below, these statements are processed in the same manner that the INCLUDE statement is processed.

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, RFALTER, 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
/dept123
/sect 456
/joe/flange.bdf'
```

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

```
rfinclude c:\project,
$ A comment line
'\Data Files' \subdir\thisfile
```

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

```
include 'MYTESTDIR:*/mytestfile.dat'
```

If the logical symbol MYTESTDIR has the value "/myfiles/test", the expanded filename is "/myfiles/test/*/mytestfile.dat". Note that this filename includes a subdirectory search request, explained below.

The following examples 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/DEPTNAME/sect 456, $ Sect Name/MYFILE.DAT”.

Requesting Subdirectory Searching

MSC Nastran has the ability to search subdirectories when attempting to locate an INCLUDE file. This can be requested in two ways:

1. Specify "*" as the *last* directory component within the *filename* specification. For example,

   ```
   include '/testdir/dir1/dir2/*/myfile.dat'
   ```

   says that the directory "/testdir/dir1/dir2" and all of its subdirectories, including nested subdirectories, are to be searched for the file to be included. Note that the "*" specification must be followed by a directory level separator character as described above and, if it is not the first character in the file name, must be preceded by a directory level separator character.
2. Omit any directory specifications on the *filename* specification and specify "*" as the *last* directory component of a directory in the directory list specified by the "jidpath" keyword. See the description of the jidpath keyword in the Keywords and Environment Variables (App. B).

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 filenames are:

1. The *filename* as specified by the include statement. If filename does not end in 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. MSC 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: Recall that character-case is insignificant to Windows file names.

For another example, consider the statement

```
include 'File1.bdf'
```

and assume "jidtype=dat" was specified or defaulted. MSC 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
```

Here is an example with a directory specification. Consider the statement

```
include 'mydir/File1.dat'
```

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

```
mydir/File1.dat
mydir/file1.dat
```

and the following filename on Windows:

```
mydir/file1.dat
```

If *filename* contains a directory component, MSC Nastran will attempt to locate one of the four LINUX or two Windows filenames in the specified directory as follows.

- If there is no subdirectory search request specified, MSC Nastran will look in the specified directory for the file.
- If there is a subdirectory search request specified, MSC Nastran will look for the file first in the specified directory then in all of its subdirectories

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, MSC Nastran will attempt to locate one of the four LINUX or two Windows filenames by searching the following directories or search paths:

- the current working directory (the "`.`" directory, i.e., the directory where the nastran command was run).
- If an RFALTER or RFINCLUDE statement is being processed
  - the directory specified by the RFADIR environment variable, if that variable is defined.
  - the directory specified by the SSSAALTERDIR environment variable, if that variable was defined.
- the directory containing the file that specified the INCLUDE, RFALTER or RFINCLUDE statement.
- If file that specified the INCLUDE, RFALTER or RFINCLUDE statement itself was included, the directory containing the parent file will be searched. This nesting will continue until the directory containing the input data file has been searched.

- the list of directories specified by the “jidpath” keyword will be searched in order, noting that one or more of the directories in this list may specify subdirectory searching.
- If an INCLUDE statement is being processed:
  - the directory specified by the RFADIR environment variable, if that variable is defined.
  - the directory specified by the SSSALTERDIR environment variable, if that variable was defined.

If no readable file can be found in any of these directories, a UFM will be issued and the job will exit.

Using the SSS Alter Library

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

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

Note:	The SSSALTERDIR specification is not required since the directory specified by the SSSALTERDIR is one of the directories automatically searched as part of INCLUDE file processing. However, using the SSSALTERDIR specification is suggested to ensure that the file in the SSS Alter directory is the one actually used instead of a file with the same name located in one of the other directories in the INCLUDE file search paths.

Resolving Abnormal Terminations

MSC 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 MSC Nastran is delivered in the file *doc_install_dir*/doc/error.doc on LINUX and *doc_install_dir*\doc\error.doc on Windows. Please consult this file for limitations and restrictions.

MSC Nastran may terminate as a result of errors detected by the operating system or by the program. If DIAG 44 is set (see the keyword diag, 180 and the MSC Nastran Quick Reference Guide), MSC 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 MSC 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 that is explained in *MSC Nastran Reference Guide*.

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.

Interpreting System Error Codes

If an operating system error occurs, an attempt is made to catch the error and place the error number in the .log file. A description of these error numbers may be obtained with the following command:

LINUX	`man 2 intro`

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 “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 MSC Nastran job (i.e., the *install_dir*/*prod_ver*/*arch*/analysis executable) and issue the command

```
kill pid
```

where *pid* is the process ID.

Job Running in the Background (after=time on Windows)

Find the job with:

```
schtasks | findstr Nast
```

Delete the job with:

```
schtasks /delete /tn JOB
```

Job Running Under LSF or PBS (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 (LINUX)

As MSC 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., MSC 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.

Linux computers support asynchronous flushing of the .f04 and .f06 files. To do this, enter the command

```
kill -USR2 pid
```

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

Common System Errors

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

LINUX Disk I/O Errors

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

  Please 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 (all platforms).

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

  MSC 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 (all systems).

  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 LINUX 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" (all others).

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.

Linux - Too many files open

Fatigue Analysis (NEF) creates DAC files from tables. In some cases, the number of DAC files may exceed a system limit. Below are instructions for increasing this limit on linux systems:

Step 1: Look at your limits:

```
#nast20231 limits
Current resource limits:
 CPU time:                          unlimited
 Virtual address space:             unlimited
 Working set size:                  unlimited
 Data segment size:                 unlimited
 Stack size:                            10240 KB
 Number of open files:                   1024 (hard limit: 1024)
 File size:                         unlimited
 Core dump file size:                       0 MB
```

Step 2: Log on as ROOT and modify /etc/security/limits.conf. The line below increases the limit to 10000 files

```
* hard nofile 10000
```

Step 3: Log out, log back in, then reset your local limits for that session:

```
ulimit -n 10000
```

Step 4: Verify new limits are available:

```
#nast20231 limits
Current resource limits:
 CPU time:                          unlimited
 Virtual address space:             unlimited
 Working set size:                  unlimited
 Data segment size:                 unlimited
 Stack size:                            10240 KB
```

```
Number of open files:              10000 (hard limit: 10000)
File size:                     unlimited
Core dump file size:                   0 MB
```

5 Using the Advanced Functions of MSC Nastran

- Overview
- Using the Advanced Keywords
- Using the NASTRAN Statement
- Managing Memory
- Managing DBsets
- Interpreting the .f04 File
- Running a Job on a Remote System
- Running Distributed Memory Parallel (DMP) Jobs
- Running with a GPGPU
- Configuring and Running SOL 700
- Running an ISHELL Program
- Using the ISHELL-INCLUDE Statement (“!”)
- Improving Network File System (NFS) Performance (LINUX)
- Creating and Attaching Alternate Delivery Databases
- Using PEM Functions in MSC Nastran
- Using Intel oneAPI Compilers with UDS

Overview

This chapter discusses the NASTRAN statement, as well as how to manage MSC Nastran's internal memory allocations and databases. It also shows how to interpret performance related information in the .f04 file and some of the lower-level database messages, 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.

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, 62; keywords specific to remote processing are listed in Running a Job on a Remote System, 103, while keywords specific to distributed processing are listed in Running Distributed Memory Parallel (DMP) Jobs, 111. Finally, a complete list of all keywords and their syntax is listed in Keywords (App. B).

All Systems

Keyword	Purpose
bpool	Specifies the number of GINO blocks set aside for buffer pooling.
buffsize	Specifies the size of database I/O transfers.
casi	If set to "no" it will disable the casi solver as a possible option when "solve=auto" is specified.
delivery	Specifies an alternate delivery database name.
dmp	Specifies the number of DMP processors used.
exe	Specifies an alternate solver executable.
ifpbuff	Specifies the size of IFPStar database I/O transfers
nastran	Specifies NASTRAN statements.
proc	Specifies an alternate solver executable file type.
rank	Specifies the rank size for the sparse solvers.
smem	Specifies the number of GINO blocks to set aside for MEMFILE portion of the SCRATCH DBset.
smp	Specifies the number of SMP processors to use in certain numeric modules.
solve	Specifies either "auto" to automatically select the solver/parallel/memory options or "train" to update Pardiso memory coefficients. These options may be placed on the command line or in the User's RC files.
sysfield	Specifies global SYS parameters. See Using the SYS Field, 91.
sys*n*	Specifies SYSTEM cell values.

LINUX Systems

Keyword	Purpose
post	Specifies LINUX commands to be executed after the job completes.
pre	Specifies LINUX 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.

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`	for MSC Nastran

An input file may contain more than one NASTRAN statement. A full description of these keywords is found in (p. 13) in the *MSC 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:

1. Default DBsets are the user-default DBsets and any DBsets specified by INIT statements (see Table 4-7).
2. Delivery DBsets contain the Structured Solution Sequences.
3. DBLOCATEd DBsets are the DBsets specified by DBLOCATE statements. See DBLOCATE (p. 85) in the *MSC Nastran Quick Reference Guide*.

BUFFPOOL, SYSTEM(114)

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

BUFFSIZE, SYSTEM(1)

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

IFPBUFF, SYSTEM(624)

See the "ifpbuff" command line keyword (ifpbuff (App. B)).

SMP, SYSTEM(107)

See the “smp” command line keyword, (smp (App. B)).

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 on all systems. See DBUPDATE (p. 92) in the *MSC Nastran Quick Reference Guide* for more information.

SYSTEM(198), SYSTEM(205)

See the “rank” keyword, (rank (App. B)).

SYSTEM(207)

See the “LOCK” SYS field keyword, (lock (App. B)).

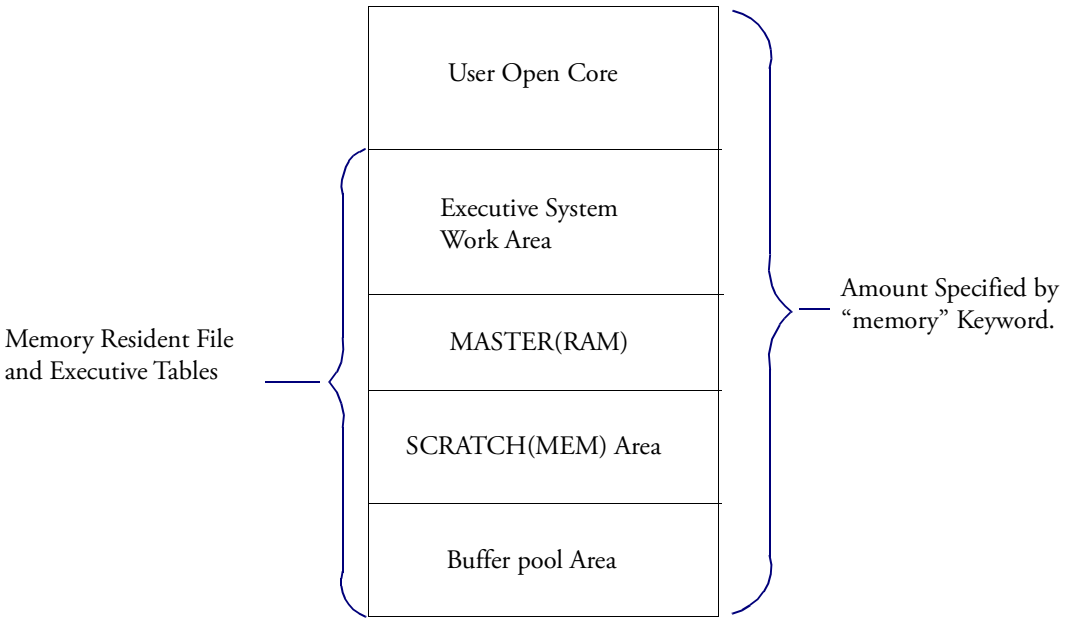
SYSTEM(275)

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.

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 MSC Nastran’s memory:



As can be seen in this diagram, the memory available for use by MSC Nastran modules (user open core) is the amount specified by the "memory" keyword (total memory) 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 X BUFFSIZE + BUFFPOOL X (BUFFSIZE + 10 ))
```

MEM	The total amount of memory specified by MEMORY keyword. The installation default is "mem=max. Set by the "memory" keyword, (p. 191). If "mem=max" is specified then the max physical is allocated. An estimate for the solver is made and the remaining memory is given to buffpool. Refer to the *MSC Nastran Release Guide* for more information.
EXEC	The executive system work area. The size is approximately 7 MB.
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 GINO Blocks of BUFFSIZEs.
BUFFSIZE	The maximum BUFFSIZE used for all the DBsets referenced by the job. The default is 32769. Set by the "buffsize" keyword, buffsize (App. B).
BUFFPOOL	The buffer pool area for permanent database files. Refer to bpool for details. If "mem=max" is specified, an estimate of memory is made and the remaining RAM is used for buffpool.

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.

nast_ver `example scratch=mini`	for MSC Nastran

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, (page 86).

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 4-7)

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 4-7)

Keyword	Purpose
buffio	Use intermediate buffers to hold database file records

Keyword	Purpose
wnum	Specifies the default number of buffers used for database files
wsize	Specifies the default size of buffers used for database files
raw	RAW=YES to specify that no intermediate buffering of file data is to be done for the next file to be opened/created. Because of restrictions imposed by the operating system on the use of this capability, “RAW=YES” will only be honored when: Buffered or asynchronous I/O is being used. That is, “BUFFIO=YES”or "ASYNC=YES" must also be specified. The record length of the file (i.e., the RECL value specified when the file is created or opened), when converted to bytes, must be an exact multiple of the disk sector size (normally 512 bytes) of the disk containing the file. If either of these conditions is not met, “RAW=NO” will be forced. RAW=NO to specify that intermediate buffering of file data is to be allowed for the next file to be opened/created. This is the default.

Systems Supporting Async I/O (see Table 4-7)

Keyword	Purpose
async	Use intermediate buffers to hold database file records, doing predictive read-ahead
wnum	Specifies the default number of buffers used for database files
wsize	Specifies the default size of buffers used for database files
raw	See note for raw keyword in Systems Supporting Buffered I/O (see Table 4-7), 91.

Using File Mapping

Notes:

- See Table 4-7 to determine if file mapping is available on your computer.

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 of its normal virtual memory management. File mapping can be used for both permanent and temporary DBsets.

The “wsize” and “wnum” parameters, described in SYS Parameter Keywords, 215, 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 globally using the “sysfield” command line keyword (page 86) and, for individual DBsets, using the ASSIGN statement SYS field or the logical-name capability of the "sysfield" command line keyword. These same statements can be used to specify the number and size of the windows using the "wnum" and "wsize" keywords.

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
```

Enabling file mapping for all but a specified set of DBsets may be done in either of the following ways:

- using both the “sysfield” keyword and ASSIGN specifications. In the command initiation file (nastran.ini), RC file, or on the command line, specify:

  ```
  sysfield=mapio=yes
  ```

  and, in the MSC Nastran data file, specify:

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

  for those files to be processed using normal disk I/O processing.
- using the logical-name capability of the "sysfield" keyword. In the command initialization file, RC file, or on the command line, specify:

  ```
  sysfield=mapio=yes,logical-name(mapio=no)
  ```

If more than one file is to be processed using normal disk I/O processing, the other logical-names may be specified on the same "sysfield" statement, on subsequent "sysfield" statements or using the "wildcard" capabilities of the logical-name capability. For example, if there are two user DBsets, USER1 and USER2, that are to be processed using normal disk I/O processing, specify:

```
sysfield=mapio=yes,user*(mapio=no)
```

Disabling file mapping for all but a specified set of DBsets may be done in either of the following ways:

- using both the “sysfield” keyword and ASSIGN specifications. In the command initialization file, RC file, or on the command line, specify:

  ```
  sysfield=mapio=no
  ```

  and, in the MSC Nastran data file, specify:

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

  for those files to be processed using file mapping.
- using the logical-name capability of the "sysfield" keyword. In the command initialization file, RC file, or on the command line, specify:

  ```
  sysfield=mapio=no,logical-name(mapio=yes)
  ```

If more than one file is to be processed using file mapping, the other logical-names may be specified on the same "sysfield" statement, on subsequent "sysfield" statements or using the "wildcard" capabilities of the logical-name capability. For example, if the scratch DBsets, SCRATCH and SCR300, are to be processed using file mapping, specify:

```
sysfield=mapio=no,scr*(mapio=yes)
```

Using Buffered I/O

Notes:

See Table 4-7 to determine if buffered I/O is available on your computer.

Buffered I/O instructs MSC 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 MSC 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 MSC Nastran. Buffered I/O can be used for both permanent and temporary DBsets.

The “wsize” and “wnum” parameters, described in SYS Parameter Keywords, 215, 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 globally using the “sysfield” command line keyword (page 86) and, for individual DBsets, using the ASSIGN statement SYS field or the logical-name capability of the "sysfield" command line keyword. These same statements can be used to specify the number and size of the buffers using the "wnum" and "wsize" keywords.

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
```

Enabling buffered I/O for all but a specified set of DBsets may be done in either of the following ways:

- using both the “sysfield” keyword and ASSIGN specifications. In the command initialization file, RC file, or on the command line, specify:

  ```
  sysfield=buffio=yes
  ```

  and, in the MSC Nastran data file, specify:

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

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

- using the logical-name capability of the "sysfield" keyword. In the command initialization file, RC file, or on the command line, specify:

  ```
  sysfield=buffio=yes,logical-name(buffio=no)
  ```

If more than one file is to be processed using normal disk I/O processing, the other logical-names may be specified on the same "sysfield" statement, on subsequent "sysfield" statements or using the "wildcard" capabilities of the logical-name capability. For example, if there are two user DBsets, USER1 and USER2, that are to be processed using normal disk I/O processing, specify:

```
sysfield=buffio=yes,user*(buffio=no)
```

Disabling buffered I/O for all but a specified set of DBsets may be done in either of the following ways:

- using both the “sysfield” keyword and ASSIGN specifications. In the command initialization file, RC file, or on the command line, specify:

  ```
  sysfield=buffio=no
  ```

  and, in the MSC Nastran data file, specify:

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

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

- using the logical-name capability of the "sysfield" keyword. In the command initialization file, RC file, or on the command line, specify:

  ```
  sysfield=buffio=no,logical-name(buffio=yes)
  ```

  If more than one file is to be processed using buffered I/O, the other logical-names may be specified on the same "sysfield" statement, on subsequent "sysfield" statements or using the "wildcard" capabilities of the logical-name capability. For example, if the scratch DBsets, SCRATCH and SCR300, are to be processed using buffered I/O, specify:

  ```
  sysfield=buffio=no,scr*(buffio=yes)
  ```

Using Asynchronous I/O

Notes:

See Table 4-7 to determine if asynchronous I/O ("Async I/O") is available on your computer.

Asynchronous I/O instructs MSC Nastran to use a predictive "read-ahead" algorithm to detect read patterns within a file (either forwards or backwards) and to issue asynchronous reads to bring data in from the file in anticipation of its later use. These reads use “buffers” or intermediate memory areas to hold the records of a file before they are actually used, i.e., copied to the MSC Nastran internal areas. The primary purpose for using asynchronous I/O is to have records already in buffers when they are requested by MSC Nastran through the use of the predictive logic. A secondary purpose, just as for 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 MSC Nastran. Asynchronous I/O can be used for both permanent and temporary DBsets.

The “wsize” and “wnum” parameters, described in SYS Parameter Keywords, 215, specify the size of the buffer to be used to hold file records and the number of such buffers to be used. Because the number of buffers affects how much actual "read-ahead" is possible, it is important that the number of available buffers be as large as possible. Typically, this value should be at least twice the number of expected different read patterns in file (some MSC Nastran operations may be accessing as many as four different portions of the scratch files at a time). More buffers allow more read-ahead and allow several I/O streams to be active in the same file.

Asynchronous I/O is controlled globally using the “sysfield” command line keyword (page 211) and, for individual DBsets, using the ASSIGN statement SYS field or the logical-name capability of the "sysfield" command line keyword. These same statements can be used to specify the number and size of the buffers using the "wnum" and "wsize" keywords.

As an example, if asynchronous 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=async=yes
```

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

```
sysfield=async=no
```

Enabling asynchronous I/O for all but a specified set of DBsets, specifying that sixteen buffers are to be used when it is enabled, may be done in either of the following ways:

- using both the “sysfield” keyword and ASSIGN specifications. In the command initialization file, RC file, or on the command line, specify:

  ```
  sysfield=async=yes,wnum=16
  ```

  and, in the MSC Nastran data file, specify:

  ```
  ASSIGN logical-name=filename, SYS='ASYNC=NO'
  ```

  for those files to be processed using normal disk I/O processing.
- using the logical-name capability of the "sysfield" keyword. In the command initialization file, RC file, or on the command line, specify:

  ```
  sysfield=async=yes,wnum=16,logical-name(async=no)
  ```

  If more than one file is to be processed using normal disk I/O processing, the other logical-names may be specified on the same "sysfield" statement, on subsequent "sysfield" statements or using the "wildcard" capabilities of the logical-name capability. For example, if there are two user DBsets, USER1 and USER2, that are to be processed using normal disk I/O processing, specify:

  ```
  sysfield=async=yes,wnum=16,user*(async=no)
  ```

Disabling asynchronous I/O for all but a specified set of DBsets and, for those DBsets, that sixteen buffers are to be used, may be done in either of the following ways:

- using both the “sysfield” keyword and ASSIGN specifications. In the command initialization file, RC file, or on the command line, specify:

  ```
  sysfield=async=no
  ```

  and, in the MSC Nastran data file, specify:

  ```
  ASSIGN logical-name=filename, SYS='ASYNC=YES,WNUM=16'
  ```

  for those files to be processed using asynchronous I/O.
- using the logical-name capability of the "sysfield" keyword. In the command initialization file, RC file, or on the command line, specify:

  ```
  sysfield=async=no,logical-name(async=yes,wnum=16)
  ```

  If more than one file is to be processed using asynchronous I/O, the other logical-names may be specified on the same "sysfield" statement, on subsequent "sysfield" statements or using the "wildcard" capabilities of the logical-name capability. For example, if the scratch DBsets, SCRATCH and SCR300, are to be processed using asynchronous I/O, specify:

  ```
  sysfield=async=no,scr*(async=yes,wnum=16)
  ```

Interpreting Database File-Locking Messages (LINUX)

All database files are locked using the operating system function "fcntl(2)". This prevents two or more MSC 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 MSC 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 WARNING - 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, 86. This affects all DBsets in the job.
- Setting SYSTEM(207) to a nonzero value using the NASTRAN statement; see Using the NASTRAN Statement, 87. 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, 91. This only affects the specific DBset (s).

Interpreting the .f04 File

MSC Nastran writes information to the .f04 file that aids in monitoring and tuning the performance of your job. An overview of the complete .f04 file can be found in Section Output Description (Ch. 7) in the *MSC Nastran Reference Guide*. This section contains more detailed explanations of selected portions of the .f04 file.

Summary of Physical File Information

This summary table describes the physical files used for the DBsets. A sample of this table, located near the top of the .f04 file, is shown below.

```
 S U M M A R Y   O F   P H Y S I C A L   F I L E   I N F O R M A T I O N

ASSIGNED PHYSICAL FILE NAME                                    RECL (BYTES)  MODE  FLAGS
-------------------------------------------------------------  ------------  ----  ---
--
/tmp/65872_57.SCRATCH                                              8192   R/W    L
/tmp/65872_57.OBJSCR                                               8192   R/W    L
/tmp/65872_57.MASTER                                               8192   R/W    L
/tmp/65872_57.DBALL                                                8192   R/W    L
/tmp/65872_57.DBALL2                                               8192   R/W    L
/tmp/65872_57.SCR300                                               8192   R/W    L
/msc20231/linux64/SSS.MASTERA                                      8192   R/O    L
/msc20231/linux64/SSS.MSCOBJ                                       8192   R/O    L

    FLAG VALUES ARE --
      F   FFIO INTERFACE USED TO PROCESS FILE
      H   HPIO INTERFACE USED TO PROCESS FILE
      L   FILE HAS BEEN LOCKED
      M   FILE MAPPING USED TO PROCESS FILE
      R   FILE BEING ACCESSED IN 'RAW' MODE

** PHYSICAL FILES LARGER THAN 2GB FILES ARE NOT SUPPORTED ON THIS PLATFORM
```

In this summary, “ASSIGNED PHYSICAL FILENAME” is the physical FILENAME with any symbols translated; “RECL” is the record length in bytes; “MODE” is the file access mode, R/W is read-write mode, R/O is read-only mode. The “FLAGS” column will contain various letters depending on the capabilities of the platform and user requests, the text below the table indicates flag values that are possible on the specific platform.

In this example, an INIT statement was used to create the DBALL DBset with two files using the logical names DBALL and DBALL2.

Below the summary is a message indicating if large files (see Using Databases, 70) are available on this platform.
Memory Map

Immediately following the “Summary of Physical File Information” is a map showing the allocation of memory. This map is also described in Managing Memory, 89.

```
** MASTER DIRECTORIES ARE LOADED IN MEMORY.
   USER OPENCORE (HICORE)          =     3804612  WORDS
   EXECUTIVE SYSTEM WORK AREA      =       78605  WORDS
   MASTER(RAM)                     =       30000  WORDS
   SCRATCH(MEM) AREA               =      204900  WORDS (     100 BUFFERS)
   BUFFER POOL AREA (GINO/EXEC)    =       76183  WORDS (      37 BUFFERS)

   TOTAL MSC Nastran MEMORY LIMIT   =     4194300  WORDS
```

In this table “USER OPENCORE” is the amount of memory available to the module for computation purposes; “EXECUTIVE SYSTEM WORK AREA” is the space reserved for the executive system; “MASTER(RAM)” is the space reserved to cache datablocks from the MASTER DBset; “SCRATCH(MEM) AREA” is the space reserved to cache datablocks from the SCRATCH and SCR300 DBsets; “BUFFER POOL AREA” is the space reserved for the buffer pool; “TOTAL MSC Nastran MEMORY LIMIT” is the total memory allocated to MSC Nastran using the “memory” keyword.

Day Log

The Day Log portion of the .f04 is a DMAP execution summary. This log, in table format, contains the vast majority of the information in the .f04. The beginning of the Day Log is shown below:

```
DAY TIME  ELAPSED  I/O MB   DEL_MB   CPU SEC  DEL_CPU   SUB_DMAP/DMAP_MODULE MESSAGES

10:32:16     0:16    13.6       .3       .8       .0    SESTATIC     20  IFPL    BEGN
10:32:16     0:16    13.7       .1       .8       .0    IFPL         29  IFP1    BEGN
10:32:16     0:16    13.7       .0       .8       .0   IFPL          39  XSORT   BEGN
```

In the Day Log, “DAY TIME” is the time of day of the entry; “ELAPSED” is the elapsed time since the start of the job; “I/O MB” is the megabytes of I/O to the databases since the start of the job; “DEL_MB” is the delta I/O since the previous entry; “CPU SEC” is the total CPU seconds since the start of the job; “DEL_CPU” is the delta CPU since the previous entry; “SUB_DMAP/DMAP_MODULE” indicates the DMAP statement being executed; and “MESSAGES” are any messages issued by the module, “BEGN” is the start of the module and “END” is the end.

Note:	
	1. If SYSTEM(84) is set to 0, the “I/O MB” column will be the number of GINO I/Os.
	2. The “I/O MB” column will be scaled by gigabytes and a “G” will be appended after each number if the value is greater than or equal to 100 000.

User Information Messages 4157 and 6439

The UIM 4157 text provides decomposition estimates upon completion on the preface of the decomposition module. This message has a counterpart, UIM 6439, which provides actual information from the completed decomposition process. These two messages are interspersed within the Day Log at each decomposition. The following example is from a sparse decomposition.

```
 *** USER INFORMATION MESSAGE 4157 (DFMSYN)
     PARAMETERS FOR SPARSE DECOMPOSITION OF DATA BLOCK  KLL      ( TYPE=RDP ) FOLLOW
                      MATRIX SIZE =     726 ROWS           NUMBER OF NONZEROES =     16926 TERMS
           NUMBER OF ZERO COLUMNS =       0        NUMBER OF ZERO DIAGONAL TERMS =       0
                 CPU TIME ESTIMATE =       0 SEC               I/O TIME ESTIMATE =       0 SEC
       MINIMUM MEMORY REQUIREMENT =      58 K WORDS             MEMORY AVAILABLE =    6978 K WORDS
     MEMORY REQR'D TO AVOID SPILL =     133 K WORDS         MEMORY USED BY BEND  =      20 K WORDS
     EST. INTEGER WORDS IN FACTOR =      48 K WORDS          EST. NONZERO TERMS =      79 K TERMS
     ESTIMATED MAXIMUM FRONT SIZE =     210 TERMS                RANK OF UPDATE =       6
 *** USER INFORMATION MESSAGE 6439 (DFMSA)
     ACTUAL MEMORY AND DISK SPACE REQUIREMENTS FOR SPARSE SYM. DECOMPOSITION
         SPARSE DECOMP MEMORY USED =     133 K WORDS           MAXIMUM FRONT SIZE =     210 TERMS
           INTEGER WORDS IN FACTOR =       8 K WORDS    NONZERO TERMS IN FACTOR =      79 K TERMS
    SPARSE DECOMP SUGGESTED MEMORY =      91 K WORDS
```

The most important elements of the UIM 4157 message are the “MINIMUM MEMORY REQUIREMENT”, which is an estimate of the user open core memory that will allow the decomposition to run, but with heavy spilling to disk. The “MEMORY REQR’D TO AVOID SPILL” will allow the decomposition to run in “in core”, i.e., without spilling to disk. These two values represent the extremes of memory requirements, the memory for optimal CPU performance is between the two. The “ESTIMATED MAXIMUM FRONT SIZE”, a function of the model, affects the memory estimates; the minimum memory is a function of the front size, and the memory to avoid spill is a function of the square of the front size. The

"NUMBER OF NONZEROES" is the size of the input matrix, multiply this value by 8 to estimate the size of the input file in bytes. The sum of "EST. INTEGER WORDS IN FACTOR" and "EST. NONZERO TERMS" is the size of the output matrix, multiply the integer value by 4 on all machines, and the nonzero value by 8 to estimate the size of the output file in bytes. The "RANK OF UPDATE" is the number of rows that will be simultaneously updated during the decomposition. This value is set by either the "rank" keyword or SYSTEM(205).

Note:	Setting SYSTEM(69)=64 will cause MSC Nastran to terminate after printing UIM 4157. This can be useful for determining a job's memory and disk space requirements.

In UIM 6439, "SPARSE DECOMP MEMORY USED" states the actual memory used in the decomposition process. Based on the execution of the module, the "SPARSE DECOMP SUGGESTED MEMORY" will result in optimal throughput performance.

Memory and Disk Usage Statistics

These tables are written after the job has completed, and indicate the maximum memory used by any sparse numerical module and the maximum disk used by any module during the job. A sample follows.

```
+---------- SPARSE SOLUTION MODULES -----------+          +------------- MAXIMUM DISK USAGE -------------+
   HIWATER                 SUB_DMAP        DMAP             HIWATER                 SUB_DMAP        DMAP
   (WORDS)    DAY_TIME       NAME         MODULE             (MB)    DAY_TIME       NAME         MODULE
    517786    04:35:44     SEKRRS     18  DCMP              15.625    04:35:48     SESTATIC   186  EXIT
```

In the left hand table, "HIWATER (WORDS)" is the maximum amount of open core used by certain sparse numerical modules; "DAY_TIME" is the time of day the module ran. "SUB_DMAP NAME" is the name of the SUBDmap; "DMAP MODULE" indicates the line number and module name that made the maximum request. Similarly, in the right hand table, "HIWATER (MB)" is the maximum amount of disk space used by any module; "DAY_TIME" is the time of day the module ran. "SUB_DMAP NAME" is the name of the SUBDmap; "DMAP MODULE" indicates the line number and module name that made the maximum request.

Database Usage Statistics

These statistics, provided in table format, summarize the I/O activity for the DBsets.

```
*** DATABASE USAGE STATISTICS ***

+---------------- LOGICAL DBSETS ----------------+   +------------------------------ DBSET FILES ------------------------------+
  DBSET      ALLOCATED  BLOCKSIZE       USED   USED     FILE      ALLOCATED  ALLOCATED    HIWATER       HIWATER  I/O TRANSFERRED
             (BLOCKS)    (WORDS)    (BLOCKS)     %                (BLOCKS)       (GB)    (BLOCKS)         (GB)             (GB)

  MASTER         5000      65536          52   1.04    MASTER        5000       2.44         152        0.074            1.449
  DBALL       2000000      65536           3   0.00    DBALL      2000000     976.56           3        0.001            0.003
  OBJSCR         5000       8192         374   7.48    OBJSCR        5000       0.31         374        0.023            0.529
  SCRATCH     4000100      65536         291   0.01   (MEMFILE        100       0.05         100        0.049            0.000)
                                                       SCRATCH    2000000     976.56       28297       13.817          209.438
                                                       SCR300     2000000     976.56        1992        0.973           24.080
                                                                                                              ==============
                                                                                                    TOTAL:           235.498
```

Where:

This statistical table contains two parallel tables. The "LOGICAL DBSETS" table lists each DBset while "DBSET FILES" tables lists the component files of the DBset. In these tables:

LOGICAL DBSETS:

- DBSET is the name of the DBset
- ALLOCATED is the MSC Nastran DBset size limit in blocks
- BLOCKSIZE is the BUFFSIZE of the DBset minus one.
- USED (BLOCKS) and USED % are the number of blocks and percent of the DBset used. For SCRATCH it is the amount used during cleanup. Refer to HIWATER for total use.

DBSET FILES:

- FILE is the file's logical name associated with the DBset to the left.
- ALLOCATED is the number of BLOCKS/Gb allocated by MSC Nastran for the file
- HIWATER is the number of BLOCKS/Gb actually used in the file.
- I/O TRANSFERRED is te amount if I/O to the file.

In this example, the MASTER and OBJSCR DBsets are each composed of one file. The MASTER, DBALL and OBJSCR are each composed of one file. The SCRATCH DBset has three components, MEMFILE, SCRATCH, and SCR300.

This table can be used to determine if the DBsets and files are appropriately sized and the amount of I/O activity associated with each file. Best elapsed time performance can be obtained if the files with the greatest activity are on different physical devices (and better yet, separate I/O controllers or busses).

Summary of Physical File I/O Activity

This summary describes the physical file I/O for each database file.

```
*** SUMMARY OF PHYSICAL FILE I/O ACTIVITY ***

ASSIGNED PHYSICAL FILENAME                                  RECL (BYTES)  READ/WRITE COUNT  MAP WSIZE (NUM)   MAP COUNT
------------------------------------------------------------ ------------  -----------  ---------------  -----------
/tmp/65872_57.SCRATCH                                           8192             1          128KB ( 4)            1
/tmp/65872_57.OBJSCR                                            8192           378          128KB ( 4)           24
/tmp/65872_57.MASTER                                            8192          1247          128KB ( 4)           11
/tmp/65872_57.DBALL                                             8192            26          128KB ( 4)            1
/tmp/65872_57.SCR300                                            8192             1          128KB ( 4)            1
/msc20231/linux64/SSS.MASTERA                                 8192           162        N/A               N/A
/msc20231/linux64/SSS.MSCOBJ                                  8192           202        N/A               N/A
```

In this summary, “ASSIGNED PHYSICAL FILENAME”, “RECL (BYTES)”, and “MAP WSIZE (NUM)” are repeated from the “Summary of Physical FIle Information” table. “READ/WRITE COUNT” is the number of GINO reads and writes that were performed on the file and “MAP COUNT” is the number of times the map window had to be remapped (these columns are only present on systems supporting mapped I/O).

This summary can be used to tune I/O performance. For mapped I/O systems, if the map count approaches the number of reads and writes, the map size and/or the number of maps should be increased. Increasing the number of maps is suggested if a module simultaneously accesses more data blocks or matrices in a file than

there are windows. Increasing the size of the windows is suggested if a file contains very large data blocks or matrices. Best elapsed time performance, with or without mapping, can be obtained if the files with the greatest activity are on different physical devices (and better yet, separate I/O controllers or busses).

Running a Job on a Remote System

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, (page 195). In the descriptions that follow, the "local" node or system is the computer you issue the nastran command on; the "remote" node or system is the computer named by the "node" keyword, i.e., the system where the MSC Nastran analysis will run.

The method used to communicate between the local and remote nodes depends on the operating system on the remote node:

- If the remote node is a LINUX system (or a similar system such as Linux), the "ssh/scp" communications programs may be used. The users who want to use "rsh/rcp" may add the following lines to <*install_dir*>/*prod_ver*/*arch*/nastran.ini:

  s.ssh=/usr/bin/rsh

  s.scp=/usr/bin/rcp

  To configure ssh/scp:

```
# Make sure there is a directory called ${HOME}/.ssh
cd $HOME
ls .ssh
#If it does not exist, create it:
   mkdir .ssh
   chmod 700 .ssh
   cd .ssh
# Execute the command:
ssh-keygen -t rsa -f id_rsa -P ''
# It is -P followed by two single quotes.  This option will create
# two files: id_rsa and id_rsa.pub.
# Copy id_rsa into a file called identity:
cp id_rsa identity
# Appened id_rsa.pub to a file called authorized_keys
cat id_rsa.pub >> authorized_keys
chmod 600 authorized_keys
# The directory $HOME/.ssh should now contain at a minimum four files
# id_rsa, id_rsa.pub, identity and authorized_keys.
```

- If the remote node is a Windows system (i.e., a system running Windows 10), the "MSCRmtCmd/MSCRmtMgr" communications programs must be used. (See Installing/Running MSCRmtMgr, 110.)

Following are some general requirements for running remote jobs:

- MSC Nastran must be properly installed on the remote system.
- The input data file must be accessible on the local host.
- INCLUDE files must be local-to, or visible-from, the remote system unless the "expjid" keyword is used (or taken by default).
- All default output files, i.e., those without ASSIGN statements, will be written to a directory accessible to the local host.
- In a restart, i.e., a job that uses an existing database, the DBsets must be local-to, or visible-from, the remote system.

If the ssh/scp communications programs (LINUX ONLY) are to be used:

- You must have "remote execution" privileges on the remote system. That is, a password must not be required to execute a remote copy (scp) or remote shell (ssh) command. See your system administrator for information on this. You can test this with the following command:

```
ssh <node> [-l <username>] date        # All others
```

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

```
ssh node1 date
```

The output from the above command should be a single line containing the current date on node1 in a format similar to:

```
Tue Jul 16 15:05:47 PDT 2002
```

If any other output is present, please 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 for the specified remote node.

If the MSCRmtCmd/MSCRmtMgr communications programs are to be used:

- The MSCRmtMgr program must be running on the remote system, either as an installed and started service or as a console mode program running in a Command Prompt window (started with the "-noservice" command line option). You can test this with the following command:

```
<install_dir>\bin\prod_ver mscrmtcmd -h <node> -i    (from Windows)
<install_dir>/bin/prod_ver MSCRmtCmd -h <node> -i    (from LINUX)
```

where "<install_dir>" is the installation directory for the local MSC Nastran installation and "<node>" is the name of the remote node. For example:

```
c:\msc\bin\prod_ver mscrmtcmd -h node1 -i     (From Windows)
/msc/bin/prod_ver MSCRmtCmd -h node1 -i       (From LINUX)
```

The output from the above command(s) should be a single line containing configuration information for node1 in a format similar to:

```
1@2@"C:/WINDOWS/system32/cmd.exe"     (If node1 is Windows)
2@2@"/bin/bsh"                        (If node1 is Linux)
```

If any other output is present or if the request fails, please 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 for the specified remote node.

Note:	Recall that, for remote LINUX nodes, remote executions do not run a "login" shell. That is, your ".profile" or ".login" script is not executed. This is true regardless of the communications programs (ssh/scp or MSCRmtCmd/MSCRmtMgr) being used.

If the node specified by the "node" keyword is the same as the local node, the "node" keyword is ignored and processing will continue as if "node" had not been defined.

The MSCRmtCmd/MSCRmtMgr communications programs may also be used when the remote node is a LINUX system. However, there are no inherent advantages over using the ssh/scp programs.

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 Keywords that specify information about the remote system's installation or that affect the actual analysis are processed on the remote system. MSC suggests that those keywords that specify information about the remote system's installation be defined in conditional Initialization or Runtime Configuration File sections based on the "node" keyword and that those keywords that specify information about the local system's installation be defined in conditional Initialization or Runtime Configuration File sections based the "s.hostname" keyword.

The following table lists some of the keywords that affect remote processing. These keywords are described in detail in Keywords, 172.

Table 5-1 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 with batch = yes. (LINUX only.)
delete	Unconditionally deletes files after job completion.
expjid	Specifies data file expansion on the local system.

Table 5-1 Remote Processing Keywords (continued)

Keyword	Purpose
lsymbol	Specifies logical symbol values to be used on the local system.
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 version or deleted.
out	Specifies an alternate output file prefix.
rcmd	Specifies the nastran command path on the remote system.
rostype	Specifies the remote node operating system type.
rrmtuse	Specifies the communication programs to be used.
rsdirectory	Specifies the directory on the remote system to contain MSC Nastran temporary files.
scratch	Specifies the DBsets are to be deleted at job completion
sdirectory	Specifies the directory on the local system to contain MSC Nastran temporary files. If "rsdirectory" is not specified, this directory will also be used on the remote system.
symbol	Specifies logical symbol values to be used on the remote system.
trans	Requests translation of the .xdb file.
username	Specifies an alternate username on the remote host.

The "sdirectory"/"rsdirectory" keywords are 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. As part of its processing, the nastran command may generate temporary files on both the local system (e.g., as part of "expjid" processing) and on the remote system (e.g., the location where temporary RC files are placed and where output files are generated). These files are placed in the "scratch" directories on the local and remote systems. If the "rsdirectory" keyword is not specified, the "sdirectory" location must be valid on both the local and remote systems. (Note that this is not possible if the systems are running different types of operating systems.) All other keywords specifying path/file names will only be scanned on the remote system and must specify path/file names appropriate for that system.

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. Reprocess the command initialization file and any RC files if any contained conditional sections. (See Resolving Duplicate Parameter Specifications, 148 for a more complete description of Command Initialization file and Runtime Configuration file processing.)

4. Determine the full pathname of the input file so that its visibility from *remotenode* can be tested.
5. If the "rmtdeny" utility, i.e., *install_dir*/*prod_ver*/*arch*/rmtdeny on LINUX and *install_dir**prod_ver**arch*\rmtdeny.exe on Windows, 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*/*prod_ver*/*arch*/rmtaccept on LINUX and *install_dir**prod_ver**arch*\rmtaccept.exe on Windows, exists and is executable, run it and examine its output. If *remotenode* **is not** listed, display an error and cancel the job.
7. Verify that *remotenode* exists and you are able to run a command on that system. This process also determines the communications programs to be used and the *remotenode* operating system type. Although the nastran command can determine this information dynamically, processing may be much faster if you specify the proper information using the "rrmtuse" and/or "rostype" keywords (for example, in an INI or RC file conditional section). The information is set as follows:
8. If the "MSCRmtCmd/MSCRmtMgr" communications programs are selected (by either defining "rrmtuse=mscrmtcmd" or defining "rostype=windows" or dynamically selected), the *remotenode* operating system type is determined automatically.
9. If the "ssh/scp" communications programs are selected (by either defining "rrmtuse=ssh" or defining "rostype=unix" or dynamically selected), the *remotenode* operating system type is assumed to be LINUX.
10. If both the local node and remote node operating system types are LINUX, create a "touch" file in the specified output file so that its visibility from *remotenode* can be tested.
11. If "rsdirectory" has not been defined or contains multiple values, set it as follows:

    a. If "rsdirectory" has been defined but contains multiple values, change its value to the first value.
12. If "sdirectory" has not been set and the local system is Windows, set "rsdirectory" to "c:\tmp" if the *remotenode* operating system is Windows and to "/tmp" otherwise.
13. If "sdirectory" has not been set and the local system is LINUX, set "rsdirectory" to "c:\tmp" if the *remotenode* operating system is Windows and to the path defined by the "TMPDIR" environment variable otherwise.
14. If "sdirectory" has been set but rsdirectory has not been set, then "rsdirectory" will be set to the first (or only) value defined by "sdirectory.
15. Ensure "scratch=no" was set if the "dbs" keyword was set.
16. If the "rcmd" keyword was specified, attempt to execute that command on *remotenode*, displaying an error and canceling 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.
17. Run the remote nastran command identified in the previous step to determine:

    If the directory specified by "rsdirectory" is valid.

    If "scratch=no" is set, if the directory specified by "dbs" is valid.
18. The numeric format of the remote system.

19. If both the local and remote modes are LINUX, the following tests are also made:

    If the input data file is visible.

    If the "touch" file is visible.

    If "expjid" was specified, if the "expjid" expand directory is visible.
20. Display an error and cancel the job if an "rsdirectory" was identified on the command line or in a local RC file, but does not exist on the remote node.
21. 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.
22. If a "touch" file was created above, delete it.

If the local system is a LINUX system, the following steps are done in a background process (possibly some time later if "batch=yes" or "after" was specified). If the local system is a Windows system, the following steps are done from within the nastran command itself.

23. Copy the input data file (or the expanded file if "expjid" processing was performed) to the remote system's scratch directory if the remote host could not see the file or if the local and remote operating system types are different.
24. Set the "out" to the remote system's scratch directory if the remote host could not see the output directory or if the local and remote operating system types are different.
25. 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.
26. Run the job on the remote node.
27. Process the "old" and "oldtypes" keywords on the local node.
28. If the output directory was not visible from the remote node or if the local and remote operating system types are different, 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.
29. Process the "append" keyword on the local node.
30. If the output directory was not visible from the remote node or if the local and remote operating system types are different and if an .xdb file was created on the remote node, run the RECEIVE program if required by the "trans" keyword or copy the .xdb file to the directory specified by the "output" keyword and delete the .xdb file from the remote node.
31. 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.

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.

```
nast_ver example node=othernode batch=no     (LINUX)
nast_ver example node=othernode              (Windows)
```

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. (Note that Windows systems do not support the use of the "batch" keyword.)

```
nast_ver example node=othernode rcmd=/some/path/bin/nast20231
```

This job will also run on "othernode" (assumed to be a LINUX system) but the path to the nastran command has been specified explicitly.

```
nast_ver example node=othernode rcmd=d:/a/path/bin/nast20231
```

This job will also run on "othernode" (assumed to be a Windows system) but the path to the nastran command has been specified explicitly. Note the use of forward slashes ('/') in the "rcmd" value. If the local system is a Windows system, either forward slash ('/') or back slash ('\') characters may be used. If the local system is a LINUX system, forward slash ('/') characters must be used or the entire rcmd specification must be enclosed in quotes to prevent the shell from interpreting the back slash ('\') characters as "escape" characters. When the rcmd specification is used on "othernode", the forward slash characters will be changed to back slash characters as needed.

```
nast_ver example node=othernode dbs=/dbs
```

This job will also run on "othernode" (assumed to be a LINUX system) 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.

```
nast_ver example node=uxsrv rsdir=/tmp sdir=/scratch
```

This example will run a job on LINUX node "uxsrv" using the nastran command in the default PATH with all scratch files on the local system residing in /scratch and all scratch files on the remote system residing in /tmp. Note that the "sdir" and "rsdir" keywords could have been set in an RCF file.

```
nast_ver example node=uxsrv rsdir=
```

This job will use the default scratch directory on "uxsrv".

```
nast_ver example node=uxsrv rsdir= rcmd=/msc/bin/nast20231
```

This job will use the nastran command /msc/bin/nast20231 on "uxsrv".

Installing/Running MSCRmtMgr

The MSCRmtMgr program provides the server-side communications support used by the MSCRmtCmd client-side program. That is, MSCRmtMgr provides functions equivalent to the LINUX rshd/rexec services. MSCRmtMgr is primarily intended for use on Windows systems.

For Windows systems, MSCRmtMgr may only be run on Windows. MSCRmtMgr may be run as a command-mode program or as a service, providing the same functionality in either case.

Running MSCRmtMgr as a Command-mode Program

This is the simplest way to run MSCRmtMgr but is the least flexible in that MSCRmtMgr must be restarted every time the operating system is restarted. In this mode, MSCRmtMgr is started in a "Command Prompt" window that is left open as long as the Windows system is to act as a server. The command is:

```
<instdir>\bin\prod_ver MSCRmtMgr -noservice
```

The "`-noservice`" operand is required and tells MSCRmtMgr that it is not to attempt to run as a Windows service program. In this mode, MSCRmtMgr will run using the authorization and access control provided by the currently logged on user. MSCRmtMgr may be terminated using `<CNTL-C>` or by using the Task Manager.

Installing and Running MSCRmtMgr as a Windows Service

The MSCRmtMgr program may also be run as a Windows Service program. Doing this requires the Microsoft Windows Resource Kit SC.exe (Services Control) utility program, available from Microsoft, and run from a command prompt. Install MSCRmtMgr as a service using the following command:

```
sc create MSCRmtMgr type= share start= auto
binpath= <instdir>\prod_ver\arch\mscrmtmgr.exe
```

where:

`type= share`	indicates that MSCRmtMgr can be shared. This option is useful if ADMIN starts the service. If the service is only needed for a specific user then you may use “own” instead of “share”.
`start= auto`	indicates that MSCRmtMgr is to be automatically started at boot time. This may also be specified as "`start= demand`".
`binpath= ...`	specifies the full path to the MSCRmtMgr program.

Note the blanks between the equal sign following the option and the actual value. These blanks are required.

Once MSCRmtMgr has been installed as a service, it may be started or stopped using the Services Administrative Tools program or using SC.exe as follows:

To start MSCRmtMgr:

```
sc start MSCRmtMgr -service -name "MSCRmtMgr"
```

To stop MSCRmtMgr:

```
sc stop MSCRmtMgr
```

If MSCRmtMgr is no longer required, it may be deleted as a service using SC.exe as follows:

```
sc delete MSCRmtMgr
```

Note that this will remove MSCRmtMgr as a service but will not actually delete the executable. It may be reinstalled as a service using the command described above.

Running Distributed Memory Parallel (DMP) Jobs

MSC Nastran offers the ability to run certain solution sequences (see the MSC Nastran Quick Reference Guide) in parallel using the Message Passing Interface (MPI), an industry-wide standard library for C and Fortran message-passing programs. MPI programs can be run on SMP computers, NUMA computers, and distributed computers.

Note:	Further information on the MPI *standard* can be obtained online at the URL **http://www.mpi-forum.org** This is not a Hexagon AB and/or its subsidiaries sites and Hexagon has no control over the site's content. Hexagon cannot guarantee the accuracy of the information on these sites and will not be liable for any misleading or incorrect information obtained from these sites.

In most cases, MSC Nastran uses the hardware vendor's MPI implementation. While this usually results in the highest performance levels, it also presents a limitation--a DMP job can only run on computers supported by the vendor's MPI package. As an example, you cannot use a mixture of Linux and Windows machines to run a single MSC Nastran DMP job.

The following table lists the hardware and software prerequisites for **every** host that will take part in running an MSC Nastran DMP job:

Table 5-2 DMP System Prerequisites

Platform	System Prerequisites	
AMD/Intel Linux	OS	Any supported version
	MPI	Intel MPI 2019 Update 9 (Default)
AMD/Intel Windows	MPI	Microsoft MPI 9.0 (Default)
	.NET Framework	

In the descriptions that follow, the "local" node is the computer you issue the nastran command on, the "parent" node is the first computer named by the "hosts" keyword, the "child" nodes are the remaining systems listed in the "hosts" list.

The following are some general comments and requirements for running MSC Nastran DMP jobs:

1. MSC Nastran must be properly installed on, or accessible to, all the hosts listed by the “hosts” keyword.
2. On both Linux and Windows the MPI programs are delivered with MSC Nastran. If users have installed MSMPI, and MSMPI_BIN is set, then those files will be used.
3. For Linux systems you must have s-command access to each system you want to access in a distributed job.

   You can test this with the following command:

```
ssh <node> [-l <username>] date  # All others
```

   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:

```
ssh node1 date
```

   The output from the above command should in a single line containing the current date on node1 in a format similar to:

```
Sun Sep 30 13:06:49 PDT 2012
```

   If any other output is present, please determine the source of the output and correct the problem. If you cannot eliminate the output, you will not be able to use the distributed execution capabilities of MSC Nastran. If ssh and scp are not available, then RSH and RCP may be used by adding s.ssh=rsh to the command line.
4. The input data file must be accessible on the local host.
5. On Windows systems, all pathnames will be converted, if necessary, to Universal Naming Convention (UNC) format for all accessibility tests. If a pathname has no equivalent UNC name, it will be considered “not accessible”. Also, if “ccsnodesmust=no” is specified, the input data file and output directory must be visible from all nodes specified by the “hosts” keyword.
6. On all systems, you must have write access to the output directory.
7. INCLUDE files must be local-to, or visible-from, each host.
8. All default output files, i.e., those without ASSIGN statements, will be written to a directory accessible to the local host.
9. The scratch directory can be a global or local file system. MSC recommends the scratch directory be local to each host, i.e., you specify per-host “sdirectory” values. See Managing Host-Database Directory Assignments in DMP Jobs, 117 for more information.
10. The pathname of the nastran command must be the same on all hosts, or on the default PATH of each host, used in the analysis.

11. For Linux systems you must have “remote execution” privileges on all the hosts listed by the “hosts” keyword. That is, a password must not be required to execute a remote copy (scp) or remote shell (ssh or remsh) command. See your system administrator for information on this.

Note:	For Linux systems recall that remote executions do not run a “login” shell. That is, your “.profile” or “.login” script is not executed.

12. For windows:
    a. For multiple hosts the MSCRmtCmd service must be started on each host (see Installing/Running MSCRmtMgr).
    b. For multiple hosts the SMPD service must be started on each host. This service may be started several ways:
       - Individual users may issue:

         ```
         SMPD -D 3
         ```

         However, multiple users may not have this service running.
       - For multiple users, it is best to start as ADMIN with:

         ```
         C:\Windows\system32>sc start MSMPILaunchSvc
         SERVICE_NAME: MSMPILaunchSvc
         TYPE               : 10  WIN32_OWN_PROCESS
         STATE              : 2  START_PENDING
         (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
         WIN32_EXIT_CODE    : 0  (0x0)
         SERVICE_EXIT_CODE  : 0  (0x0)
         CHECKPOINT         : 0x0
         WAIT_HINT          : 0x7d0
         PID                : 9916
         FLAGS              :
         ```

    c. Multiple hosts users may get messages to allow analysis and MSCRmtMgr to run on each system the first time it occurs.

When running a DMP job, nastran keywords are processed on both the local and parent/child systems. Keywords that control the job's output and interaction with you are processed on the local system. These are:

Table 5-3 DMP Processing Keywords

Keyword	Purpose
append	Requests the .f06, .f04, and .log files to be concatenated.
childout	Specifies the .f04 and .f06 files from the child tasks are to be appended to the .f04 and .f06 files of the parent task.
hostovercommit	Requests more tasks per host than CPUs.
hosts	Specifies list of hosts to use. Separate hosts with a comma or with the PATH separator, i.e., ":" on Linux and ";" on Windows.
mergeresults	Specifies the results from each DMP task are to be merged into the standard files from the parent host.
ncmd	Linux: Specifies an alternate notification command.
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 parent/child systems.
rmtdiskmsg	Enables or disables the "sdir and/or dbs disks are remotely mounted" message.
scratch	Specifies the database DBsets are to be deleted at job completion.
sdirectory	Specifies each per-host directory to contain MSC Nastran temporary files. Separate directories with a comma or with the PATH separator, i.e., ":" on Linux and ";" on Windows.

The "sdirectory" keyword is special, as the command line, RC files on the current host, and RC files on the each parent and child host will all be considered when establishing a scratch directory. All remaining keywords are only scanned on the parent and child systems.

Once "dmparallel=*number*" 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 the parent and each child host can be tested. For Windows, this full pathname will be converted to UNC format, if necessary.
4. Create a "touch" file in the specified output directory so that its visibility from the parent and each child host can be tested.
5. If the "dmpdeny" utility, i.e., *install_dir*/*prod_ver*/*arch*/dmpdeny, exists and is executable, run it, and save its output.

6. If the "dmpaccept" utility, i.e., *install_dir*/*prod_ver*/*arch*/dmpaccept, exists and is executable, run it, and save its output.
7. Ensure "scratch=no" was set if the "dbs" keyword was set.
8. Determine every possible pairing of host and sdirectory by scanning each list in a round-robin order. That is, the first *host* is paired with the first *sdirectory*, the second *host* with the second *sdirectory*, and so on.
9. Execute the following steps for each *host-sdirectory* pair determined above until *host-sdirectory* pairs have been assigned to each of the tasks requested by the "dmparallel" keyword or no more *host-sdirectory* pairs are available. Steps a. through g. are executed only once per *host-sdirectory* pair.
    a. Verify that *host* exists. For Linux systems, verify that you are able to run a command on that system.
    b. If the "rcmd" keyword was specified, attempt to execute that command on *host*, display an error and cancel the job if it fails.
    c. Otherwise attempt to execute the pathname of the current nastran command on *host*. If it fails, attempt to execute the basename of the current nastran command on *host*. Display an error and cancel the job if both checks fail. For Windows systems, these pathnames are converted to UNC format, if necessary, before they are used.
    d. 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 *host*; and finally the numeric format of *host*.
    e. Drop this *host-sdirectory* pair from further consideration if a scratch directory was identified on the command line or in a local RC file and that specified a list of directories, but the sdirectory does not exist on *host*.
    f. Display an error and cancel the job if the numeric format of *host* differs from the numeric format of the local host or if the operating system type of *host* differs from the operating system type of the local host, i.e., you cannot mix Linux hosts with Windows hosts.
    g. Display an error and cancel the job if the directory specified by a "dbs" keyword on the command line or in a local RC file does not exist on *host*.
    h. Assign the current host-sdirectory pair to the next task; save the "sdirectory" value and the per-host visibility flags and "rcmd" value.
10. Display an error and cancel the job if one or more of the tasks requested by the "dmparallel" keyword have not been assigned.
11. Delete the "touch" file created above.
12. For Linux systems, the remaining steps are done in a background process (at possibly some time later) if "batch=yes" or "after" was specified.
    a. Copy the input data file to the scratch directory of any host that could not see the input data file.
    b. Set "out" to the host-specific scratch directory value of every host that 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 "out" directory. Copy this RC file to the host-specific scratch directory on any host that could not see the output directory.

d. Process the "old" and "oldtypes" keywords on the local node.

e. Run the DMP job using the system's MPI startup command. Note that each task will write to task-specific names.

f. If the parent task could not see the output directory, copy the output files (.f04, .f06, .log, .ndb, .pch, .plt) from the parent node to the output directory (the directory specified by the "output" keyword) and delete the files from the parent node.

g. Process the "append" keyword on the local node.

h. For Linux systems, process the "notify" keyword on the local node.

Once the job has completed, the .f06, .f04, .log, .ndb, .op2, .plt, .pch, and .xdb files from the parent task will be present as if the job were run locally.

Note:	No attempt is made to copy DBset files between the local and parent/child systems. If this is required, you must handle this yourself and set the "dbs" keyword appropriately.

Determining Hosts Used by DMP Jobs

The nastran command uses the following hierarchy to determine the list of hosts to use:

1. The nastran command "hosts" keyword on the command line
2. The nastran command "hosts" keyword in an RC file.
3. The local host.

Consider the following examples:

```
nast_ver example dmparallel=4
```

On linux, this job will run on the local host.

```
nast_ver example dmparallel=4 hosts=node1:node2:node3:node4:node5
```

On windows, this job will run on the local host.

```
nast_ver example dmparallel=4 hosts=node1;node2;node3;node4;node5
```

This job will run on the first four available nodes from the set "node1", "node2", "node3", "node4", "node5".

```
nast_ver example dmparallel=4 hosts=my.host.list
```

This job will read the file "my.host.list".

The nastran command provides a simple host allocation method. If a host listed by the "hosts" keyword is unavailable, it will be skipped and the next host considered. As long as at least the number of processors specified by the "dmparallel" keyword are available on one or more of the listed hosts, the job will be allowed to run.

Note:	You may also need to modify the <*install_dir*>/*prod_ver*/arch/nastran.dmp file if job queuing information must be embedded in the job stream. A hypothetical example is included. THE SAMPLE QUEUING INFORMATION MAY NOT WORK WITH YOUR SITE'S QUEUING REQUIREMENTS

nastran Command "hosts" Keyword (Distributed Jobs Under LSF)

The "hosts" keyword will default to the value set by LSF when running as a distributed job and no other value for "hosts" was set on the command line or in an RC file.

Example:

```
bsub -n 4 nast_ver example dmp=4
```

This job will use four hosts selected by LSF. Note, the number of tasks appears twice: once for use by LSF, and once for use by MSC Nastran.

Managing Host-Database Directory Assignments in DMP Jobs

The performance of the disk subsystem containing the permanent and SCRATCH DBsets can have a significant impact on MSC Nastran performance. In the case of a DMP job, the impact can be even greater if multiple tasks are using the same file system. For SCRATCH DBsets, there are two ways in which this can be handled: one by specifying host-specific scratch directory values in an RC file and one by specifying a list of scratch directories using the "sdirectory" keyword. For DBsets, you may specify a list of DBset locations using the "dbs" keyword. When the list method is used to specify multiple scratch directories or DBsets, the entries are paired with the "hosts" keyword specified host names in a round-robin order. With these capabilities, you can finely control the use of disk I/O access paths by your job.

In the case of SCRATCH DBsets, the simplest method of specifying individual directory paths for each host is to use the RC file method, reserving the "sdirectory" list method for situations where you are assigning multiple DMP tasks to a single host and you want to separate the SCRATCH DBsets, placing each on a separate file system. The following is an example of an RC file that defines the default SCRATCH directory for each node in a four-node configuration with node names "node1", node2", "node3" and "node4". This example assumes that the MSC Nastran installation directory is available to and is used by each node. The following would, typically, be included in the RC file in the "conf" directory, noting that the technique is available on all platforms, where customizing the node-specific SCRATCH directory pathnames being the only change needed:

```
# Define node-specific scratch directories
[ s.hostname = node1 ]
sdir=/node1/scratch

[ s.hostname = node2 ]
sdir=/node2/scratch

[ s.hostname = node3 ]
sdir=/node3/scratch

[ s.hostname = node4 ]
sdir=/node4/scratch
```

Note that none of the “sdirectory” keyword values should be in “list” format, that is, contain multiple directories separated by a comma or colon (Linux) or semi-colon (Windows) unless you wish that specification to be used whenever DMP processing is requested and when you are sure that the list will match the order in which host names are specified in the “hosts” keyword.

The following examples show the effect of the round-robin ordering. These examples are the Linux syntax.

```
nast_ver example dmparallel=4 hosts=a:b sdirectory=/aa:/ba:/ab:/
bb dbs=/aa:/ba:/ab:/bb
```

This example will assign the following host-sdirectory pairs (assuming hosts “a” and “b” each have at least two processors):

Task	Host	Scratch Directory	DBS Directory
1	a	/aa	/aa
2	b	/ba	/ba
3	a	/ab	/ab
4	b	/bb	/bb

If directory “/ba” was not available for writing by you on host “b”, the tasks assignments would be (assuming host “a” has at least three processors):

Task	Host	Scratch Directory	DBS Directory
1	a	/aa	/aa
2	a	/ab	/ab
3	b	/bb	/bb
4	a	/aa	/aa

Managing Files in DMP Jobs

When an MSC Nastran DMP job is running, the input file is directly read by each MPI task that can read the file, e.g., via NFS. Each host that cannot read the input file will read a local copy of the file that is copied, via rcp(1) or scp(1) for Linux or using MSC developed utilities on Windows, to the job's scratch directory ("sdirectory" keyword) before the job begins.

A similar check is made for the output directory. Any host that can write to the output directory ("out" keyword) will directly write its .f04, .f06, .log and other default output files to that directory. Any host that cannot see the output directory will write its default output files to the job's scratch directory. For Linux systems, these files will then be copied, again via rcp(1) or scp(1) for Linux, back to the output directory at the end of the job.

Note:	The nastran command will perform these tests by converting your pathname value to an absolute pathname. As a result, a path that varies depending upon the host will be labeled as unreadable.

If the "sdirectory" keyword is not specified on the command line or in an RC file on the local host or is not specified in list format, i.e., with multiple directory specifications separated by commas or colons (Linux), each parent or child host will use its own scratch directory. This directory is determined on the parent and each child host by examining its command initialization file and version-specific RC files if the "version" keyword was defined.

DO NOT use an ASSIGN statement for any file that will be written by MSC Nastran in a Distributed Memory Parallel (DMP) job. Instead, use the "sdirectory" and "dbs" keywords to specify names of the SCRATCH and permanent DB Sets.

DMP Performance Issues

In addition to the normal performance issues associated with a serial or SMP job, a DMP job adds communication bandwidth as a critical performance characteristic. The basic communications channels, are:

- Shared memory - SMP and NUMA systems.
- Interconnect, adapter, or switch - NUMA and distributed systems.
- High-speed special-purpose network.
- TCP/IP network - all systems.
- Infiniband works on Linux with IntelMPI. To use it, please add the following to your command line (note that performance gained from Infiniband varies:

  ```
  symbol=I_MPI_DEVICE=rdssm
  ```

The performance of any MSC Nastran job is very much dependent on CPU, memory subsystem, and I/O subsystem performance. A Distributed Memory Parallel (DMP) job on an SMP or NUMA system is extremely sensitive to I/O subsystem performance since each task independently accesses the I/O subsystem.

You are especially encouraged on SMP and NUMA systems to partition your scratch directory and database assignments on DMP jobs using the "sdirectory" and "dbs" nastran command keywords.

Example:

```
nast_ver example dmp=4 sdir=/scr1:/scr2:/scr3:/scr4\
dbs=/dbs1:/dbs2:/dbs3:/dbs4
```

The following assignments will be made in this job:

Task	sdirectory	dbs
1	/scr1	/dbs1
2	/scr2	/dbs2
3	/scr3	/dbs3
4	/scr4	/dbs4

The preceding example will perform substantially better than the following job, which uses the default assignments for the “sdirectory” and the “dbs” keywords.

Example:

```
nast_ver example dmp=4
```

While the ultimate effect of the communications channel on job performance is dependent upon the solution sequence, for best overall job performance, you should try to use the fastest communications channels available.

Running with a GPGPU

MSC Nastran can now utilize Nvidia CUDA-capable GPGPU (General-Purpose computational Graphics Processing Units) cards as well as CPUs. GPGPU support for AMD and/or other computational modules will follow in future MSC Nastran releases.

System Requirements

- Nvidia CUDA-capable GPGPU card(s) with at least 1.5GB on-board memory
- A double precision card (e.g. A100)
- MSC Nastran GPGPU license
- The Window's driver needed in past releases for the GPU was 396.26. The new driver needed is:

  ```
  NVIDIA-SMI 465.19.01      Driver Version: 465.19.01
  ```
- The minimum driver required on Linux is:

  ```
  NVIDIA-SMI 471.41      Driver Version: 471.41
  ```
- MSC has tested the following Nvidia GPU cards:
  - Quadro GV100, Tesla K40m, P40, V100 and A100

The Nvidia M4000 will not work.

GPGPU card Identification

If a GPU card cannot be detected by Nastran, change to the TCC (Tesla Compute Cluster) mode with the Nvidia SMI utility. Refer to the pertinent Nvidia documents for details.

On linux, "nvidia-smi" may be used to determine your cards:

```
Node <80> /usr/bin/nvidia-smi
+-----------------------------------------------------------+
| NVIDIA-SMI 471.41   Driver Version: 471.41         |
|-------------------------------+----------------------+----------------------+
| Nb.  Name                     | Bus Id        Disp.  | Volatile ECC SB / DB |
| Fan   Temp   Power Usage /Cap | Memory Usage         | GPU Util. Compute M. |
|===============================+======================+======================|
| 0.  Tesla C2075               | 0000:0F:00.0  Off    |          0          0 |
|  30%   59 C  P0    79W / 225W |   1%   79MB / 5375MB |    1%     Default    |
|-------------------------------+----------------------+----------------------|
| 1.  Tesla C2070               | 0000:42:00.0  On     |          0          0 |
|  32%   83 C  P0    N/A /  N/A |   2%   89MB / 5375MB |    2%     Default    |
|-------------------------------+----------------------+----------------------|
| Compute processes:                                          GPU Memory |
|  GPU  PID     Process name                                  Usage      |
|=============================================================================|
|  0.  5996     /devl/msc/2023.1/msc20231/linux64/analysis             66MB  |
|  1.  6043     /devl/msc/2023.1/msc20231/linux64/analysis             66MB  |
+-----------------------------------------------------------------------------+
```

On Windows, to verify TCC, one has to run as administrator as follows:

1. In the 'startup' window, type in 'cmd' and hit 'ctrl+shift+enter' instead of just 'enter'
2. In the 'cmd' window, change dir to where nvidia-smi.exe is and execute the command:

```
C:\Program Files\NVIDIA Corporation\NVSMI>nvidia-smi --id=1 --driver-model=1
```

Driver model is already set to TCC for GPU 0:42:0.

If one doesn't run as root, one would not be able to determine the mode:

```
C:\Program Files\NVIDIA Corporation\NVSMI>nvidia-smi --id=1 --driver-model=1
Unable to determine current driver model for GPU 0:42:0: Unknown Error
```

Using the GPU

Submit with gpuid, e.g,

```
nast_ver job gpuid=0
```

In MSC Nastran 2023.1 release, the GPUs are used for two types of operations: matrix factorization and matrix multiplication. In the routines that use GPUs for matrix factorization, only one GPU will be used per DMP process. So in order to use multiple GPUs, the user must also use multiple DMP processes:

```
nast_ver job gpuid=0,1,...,ngpu-1 dmp=ndmp,  where ndmp ≥ ngpu
```

Each DMP process will be assigned a GPU ID in round robin fashion.

In MPYAD and FASTFR modules, on the other hand, multiple GPUs can be used for matrix multiplication by a single process, as long as the number of SMP-threads is equal to or larger than the number of GPUs. These modules also use multiple parallel streams for data transfer between the GPU and the host system, so it is recommended to use the maximum number of available CPU cores/threads because that helps reduce the overhead of data transfer between the host and the GPUs.

So, for example, in order to achieve the best performance on a system with 20 CPU cores and 2 GPUs the user is advised to run Nastran with:

```
nast_ver job dmp=2 smp=10 gpuid=0,1
```

or

```
nast_ver job dmp=4 smp=5 gpuid=0,1
```

Guidelines and Limitations

For the best performance, a sparse direct solver intensive SOL101 or SOL108 job should set sys653=3, or sys653=1 if the model is positive definite or diagonally dominant.

Known Issues

The pivoting method in sys653=3 is not as robust as that in sys653=1. Therefore, for models that have large numbers of Lagrange multipliers, such as those encountered in SOL400 and SOL200, sys653=3 should be avoided. Generally speaking, only SOL101 and SOL108 jobs should use sys653=3 until future improvement.

Verification GPGPU is Being Activated

The log file will have lines similar to:

```
07:50:58  Acquired 1 license for Nastran GPGPU from license server on host msclic
…
device id = 0
device id = 1
```

For more information on GPGPU, see http://gpgpu.org.

Configuring and Running SOL 700

Hardware and Software Requirements

Although no specific hardware requirements exist for MSC Nastran to run distributed memory parallel mode, it is preferable to have fast network connections between the machines if more than one machine is used. It is recommended that the network should have a speed of at least 100 MBit per second. If only two machines are to be used, you can use a hub or a cross-over cable to connect them. If more than two machines are to be used, a switch is preferable. TCP/IP is used for communications.

Compatibility

MSC Nastran supports connection of homogeneous networks with machines of the same type. Two machines are compatible if they can both use the same executables. Some examples of compatible machines are:

- Several machines with exactly the same processor type and O/S.

Definitions

1. Root machine: The machine on which the job is started.
2. Remote machine: Any machine other than the root machine that is part of a distributed parallel run on the network.
3. Shared installation: MSC Nastran is installed in an NFS shared directory on one machine only. Other machines can access the executables since the directory is shared.
4. Distributed installation: MSC Nastran is installed on all machines. Each machine accesses its own versions of the executables.
5. Distributed execution: SOL 700 is run on multiple machines that are connected with a network. Each machine loads the executables either from shared or local directories and then executes them.
6. Shared I/O: MSC Nastran reads and writes data in an NFS shared directory. Each executable running on the network reads and writes to the same directory.
7. NFS – Network File System.

Network Configuration

MSC Nastran only needs to be installed on the root machine where the installation directory is shared via NFS (shared installation). It can also be installed on the remote Machines, which then use their own executables (distributed installation). The root machine is the one on which the SOL 700 job is started. The remote machines can be located anywhere as long as they are connected to the network. The working directory on each machine can be a shared directory on any machine on the network (shared I/O) or it

can be a local directory on the hard disk of each machine in the analysis (distributed I/O). User Notes in this chapter provide instructions for specifying the working directory to use.

Installation Notes

This part describes the specific steps needed to install and set up a network version of SOL 700. For distributed parallel, install MSC Nastran on the root machine and, if needed, on the remote machines. MSC Nastran only needs to be installed on the root machine if it is a shared installation. There is nothing special that needs to be done related to the installation itself for the network version. In order to run parallel jobs on machines connected over the network, jobs have to be set up properly. If any of the remote hosts do not have MSC Nastran installed, the installation directory on the root machine needs to be shared using NFS or some other mechanism so that all executables are available from the remote machines. **Users need to be able to connect between the machines using rlogin without having to provide a password.** For some

platforms, special attention requiring root access are required to make SOL 700 jobs run which will be described in the next section.

Platform Specific MPI Configurations

Linux X8664 (Intel MPI)

In order to run Nastran SOL 700 on a Linux X8664 machine with Intel MPI, Python Version 2.7 or higher needs to be installed and must be the default. When Python version 2.6 is the default, SOL 700 with Intel MPI, may not be able to execute properly.

User Notes

This section assumes that MSC Nastran, including the MPI, has been successfully installed on two machines that are to be used in a distributed analysis and that the appropriate MSC licenses are in order.

Assume that host1 is the host name of the root machine from which the job is to be started and the host name of the other machine (the remote machine) is host2.

First, make sure that the two machines are properly connected. (On Windows: from host1, access host2 with Network Neighborhood.) If this is not possible, a network run will not be possible.

Command Line Option

SOL 700 may be run parallel in a manner similar to standard DMP parallel. If a user specifies dmp700=n (where n is the number of processors) and does not have a sol700.pth file, then a temporary file is created using the hosts specified by HOSTS= either in the command line or RC file. The sol700.pth file is used if PATH=3 is specified on the SOL 700 entry.

PATH=3 Option

In order to perform an analysis over a network, a special file called a *hostfile* needs to be created by the user. This file defines which machines are to be used, how many processes are to run on each, what working directory should be used, and where the Dytran executable can be found on each machine. No specific name or extension is used for the host file except that the name *jobid.hostfile* must be avoided since it is used internally.

Specification of the Hostfile

The hostfile has the following general format:

```
host1 n1 workdir1 exe1
host2 n2 workdir2 exe2
host3 n3 workdir3 exe3
```

Each line must start at column 1 (no initial blanks). Blank lines and lines beginning with a # (number symbol) are ignored. The first entry is the host name of a machine to be used in the analysis. The root machine must be listed first and each machine must only occur once. The second entry specifies the number of processes to run on the machine specified in the first entry. The default is 1 cpu for each machine. The

sum of the number of processes given in the hostfile will be equal the number of domains used. In a five-domain job, it is will be n1+n2+n3=5.

The third entry specifies the working directory to use on this host. This entry is ignored for SOL 700, as all I/O occurs on the working directory of the root machine.

The fourth entry specifies the location of the executable including to the full path to the executable. For the first entry (the root machine), the MSC Nastran Installation can figure out automatically which exe to take. The default for all subnodes is the same location as the location of the exe on the root machine. In case the subnodes have a different MSC Nastran installation location, this can be specified here. In case `exe2`, `exe3`, etc. are used, `n2`, `n3` and `workdir2`, `workdir3` are required input and can not be skipped. In case the location of the executables on the remote machines is exactly the same as on the root machine, the workdir and the exe location can be omitted from the hostfile.

Example of the Hostfile

The different domains are associated with the different machines as follows. Suppose a five-domain job is run using a hostfile, defined as:

```
host1 2
host2 1
host3 2
```

Domains 1 and 2 will be associated with host1, domain3 with host2 and domains 4 and 5 with host3.

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. MSC 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

File-Type	Command Processor
.pl	perl
.prl	perl
.py	python

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
.py	python

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, MSC Nastran will resort to the second feature design to assist in using the ISHELL module, the “ishellpath” keyword. If this keyword is set, MSC 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 MSC_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 *doc_install_dir*/tpl6/cc705/qaishell.dat, *doc_install_dir*/tpl/qaishell and *doc_install_dir*/tpl/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.

```
nast_ver 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, qaishell.prl, QAISHELL.py and finally qaishell.py. 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, QAISHELL.PRL and finally QAISHELL.PY. 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 ".ksh" 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.

You may need to define "CMD.EXE" on Windows as the command processor for certain ".exe" files. Examples include 16-bit compiled Basic programs.

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.

Using the ISHELL-INCLUDE Statement ("!")

The ISHELL module provides a way to dynamically alter the instruction stream of a running DMAP, making it easier to integrate your own programs, and simplifying the task of customizing MSC Nastran. The ISHELL-INCLUDE statement ("!") extends the ISHELL feature to the instruction stream of the input file. This capability is derived by merging the features of both the ISHELL and the INCLUDE statements (by first executing the external program and then including the output in the input stream). The format is:

! *embedded shell command.*

! *continuations are indicated simply by the presence*

! *of another "!" in the first non-blank position of the next line.*

! *all characters following the "!" are passed to the appropriate*

! *shell for evaluation.*

The *shell* (or command processor) is determined by the MSC_ISHELLEXT environment variable, or by the *ishellext* keyword from the command line or RC file (see "Running an ISHELL Program" for more details). On LINUX systems, the command processor associated with the *null* file type is used for the ISHELL-INCLUDE statement. In most cases this requires one of the following keyword assignments to be added to the command line:

ishellext=.=/bin/csh	# for csh scripts
ishellext=.=/bin/ksh	# for ksh scripts
ishellext=.=/bin/sh	# for sh scripts
ishellext=.=perl	# for perl scripts
ishellext=.=python	# for python scripts

Note: The ISHELL-INCLUDE statement is currently not supported for Windows.

Like the INCLUDE statement, the ISHELL-INCLUDE statement can appear anywhere in the input file. However, the output (captured from "stdout") must be appropriate to the section in which it will be included (i.e. the final input stream must constitute a valid MSC Nastran input file). Unlike the INCLUDE statement, nested ISHELL-INCLUDE statements are not supported.

The processing of an embedded shell script is done as follows:

1. The entire script is extracted and written to a temporary file.
2. If the ISHELL-INCLUDE occurs within a DMAP alter, the processing is delayed until the DMAP compiler is invoked.
3. Otherwise, the input file processing is suspended, and the external program is executed. Output from the external program is captured to another temporary file which is immediately opened and included into the input stream.
4. Once the reading of the entire output is completed, processing of the input file is resumed.

The following additional processing steps are done for an embedded shell script located within a DMAP alter:

1. The DMAP statements that are selected by the alter are extracted to an external file named: "ishell.stdin".
2. If stdout is written to, then that output is included in the alter; otherwise, "ishell.stdin" is read. This allows an interactive program like "vi" to simply save the modified input buffer, and it is automatically included in the alter.

An immediate benefit of the ISHELL-INCLUDE statement is the ability to customize the MSC Nastran job to dynamically record (and/or respond) to the run time environment. The following example captures the value of a few environment variables as comments in the f06 file:

```
example.dat:
        echooff                 $ removes copy of the ishell script below from
                                the f06
         ! echo                 # just the results from the shell will be
        "echoon"                echoed
        ! echo "$"
        ! echo "$    License File: `printenv MSC_LICENSE_FILE`"
        ! echo "$  Job was run on host: `printenv HOST`"
        ! echo "$    Nastran Version: `printenv MSC_VERSD`"
        ! echo "$  Temporary Directory: `printenv MSC_SDIR`"
        ! echo "$      TMPDIR: `printenv TMPDIR`"
        !echo "$       Scratch: `printenv MSC_SCR`"
        !echo "$          User: `printenv USER`"
        !echo "$       Display: `printenv DISPLAY`"
        !echo "$          Base: `printenv MSC_BASE`"
        !echo "$          Path: `printenv MSC_JID`"
        !echo "$       Memory: `printenv MSC_MEM`"
        !echo "$     Assign File: `printenv MSC_ASG`"
        !echo "$          Shell: `printenv SHELL`"
        !echo "$     Ishell Ext: `printenv MSC_ISHELLEXT`"
        !echo "$     Ishell Path: `printenv MSC_ISHELLPATH`"
        !echo "$      Ishell File: $0"
        ! echo "$ "
```

The example above should be executed with /bin/csh as the command processor:

```
> nastran example.dat scr=yes ishellext=.=/bin/csh
```

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 MSC 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 MSC Nastran jobs accessing disks on remote computers via NFS, Hexagon recommends increasing 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.

Creating and Attaching Alternate Delivery Databases

MSC Nastran uses the Structured Solution Sequences (SSS), located in *install_dir*/*prod_ver*/*arch* on LINUX and *install_dir**prod_ver**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*/*prod_ver*/nast/del/ directory on LINUX and *install_dir**prod_ver*\nast\del\ on Windows:

Filename	Description
buildsss.py	Python script used to build the delivery data base
*.dmap	SubDMAP source
*.dti	timing constants

Using MSC-Supplied Source

To rebuild the delivery database using the MSC-supplied source, the following procedure is used:

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

   On Linux:

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

   Or on Windows:

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

2. Rebuild the delivery database.

   On Linux:

```
$MSC_BASE/$MSC_ARCH/nast/del/buildsss.py -prog
$MSC_BASE/bin/nast20231
```

   Or on Windows

```
$MSC_BASE\$MSC_ARCH\nast\del\buildsss.py -prog
$MSC_BASE\bin\nast20231
```

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, (p. 180).

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

on LINUX

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

Or on Windows.

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

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,

On LINUX,

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

   Or on Windows.

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

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

   On LINUX,

```
cp install_dir/prod_ver/nast/del/subDMAP.dmap .
cp install_dir/prod_ver/nast/del/nddl.ddl .
```

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

```
copy install_dir\prod_ver\nast\del\subDMAP.dmap .
copy install_dir\prod_ver\nast\del\nddl.ddl .
```

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

   prod_ver `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 (page 180).

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

On LINUX

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

or on Windows.

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

Using PEM Functions in MSC Nastran

Running PEM jobs with multiple hosts on LINUX systems

For Porous-elastic Material, PEM, job with multiple processors from single host, it is sufficient to specify DMP and/or SMP during job submittal. To run PEM job with multiple hosts, following steps need to be observed.

1. PEM job with multi-hosts must be submitted from a directory which is visible across all hosts.
2. A local rc file must be prepared with following line

   ```
   j.env=ACTRAN_MPI_OPTS='-machinefile hostfile -mca plm_rsh_agent
   /usr/bin/ssh'
   ```
3. A 'hostfile' must exist in the directory where job will be submitted. This 'hostfile' will be used by both MSC Nastran and ACTRAN.
4. 'hosts=hostfile' should be used as part of PEM job submittal command.

Running PEM jobs with multiple processors on Windows systems

To run a PEM job with DMP>1 and/or SMP>1 on Windows system, MSMPI_BIN environment variable must be set as follows

```
(set MSMPI_BIN="c:\Program Files\Microsoft MPI\bin")
```

Or, use control panel (system/advanced system settings/environment variables/system variables) to set MSMPI_BIN.

If PEM DMP>1 and/or SMP>1 job hangs, execute following command at command prompt:

```
install_dir\prod_ver\actran\win64\Actran_20nn\mpi\intelmpi\
bin\hydra_service.exe -install
```

Using Intel oneAPI Compilers with UDS

MSC Nastran allows users to recompile files. This capability is accessed with the `uds` keyword. The compilation has required the same Intel compilers (linux: `FORTRAN/C`, Windows: `FORTRAN`) that MSC used for development. MSC now allows the Intel oneAPI compilers to be used.

To access Intel oneAPI compilers, add `oneAPI=true` to the command line.

Details/Caveats:

- This option will only affect FORTRAN and not C compilers on windows.
- If Intel oneAPI compilers are not available, but the standard compilers are available, then a warning will be printed and the job will continue using the standard compilers.
- If UDS jobs are required to build IDL files, then in order to use the Intel oneAPI compilers, the `SDK_InstallDir/Tools/genskeleton.py` file needs to be modified by adding `STRICT_CHECK=False` to the cmd around line 224:

```
else:
    if len(buildargs):
    cmd += " " + " ".join(buildargs)

    cmd += " STRICT_CHECK=False"

    if self._VERBOSE: print(cmd)
    pcmd = Popen(cmd, shell=True, stdout=PIPE, stderr=STDOUT, encoding='utf-8')
    output = pcmd.communicate()[0].splitlines()

    builderror = None
    builddone = None
```

- The scripts were tested with IntelONEAPI compilers/versions installed here

  # LINUX

  `\opt\intel\oneAPI\compiler\2022.0.1\`

  # WINDOWS

  `C:\Program Files (x86)\Intel\oneAPI\compiler\2022.0.0`

  If the user has them installed in a different place, then modify:

  `MSC_BASE\msc20231\sdk\bin\uds.py`

 # WINDOWS

```
1629     if (ONEAPI):
1630           INTEL_COMP_ROOT=os.path.join("C:\\","Program Files
(x86)","Intel","oneAPI","compiler","2022.0.0","windows")
1631           args.append('STRICT_CHECK=False')
1632           args.append('INTEL_COMP_ROOT='+INTEL_COMP_ROOT)
1633           args.append('FORTRAN='+INTEL_COMP_ROOT+'/bin/intel64/ifort')
1634     else:
1635           args.append('scons')
1636           if (ONEAPI):
1637             INTEL_COMP_ROOT="/opt/intel/oneapi/compiler/2022.0.1/linux"
1638             args.append('STRICT_CHECK=False')
1639             args.append('INTEL_COMP_ROOT='+INTEL_COMP_ROOT)
```

A Configuring the Runtime Environment

- Specifying Parameters
- User-Defined Keywords
- Resolving Duplicate Parameter Specifications
- Customizing Command Initialization and Runtime Configuration Files
- Symbolic Substitution

Specifying Parameters

MSC Nastran execution is controlled by a variety of parameters, either keywords or special Nastran statements, both required and optional. The purpose of this section is to describe how and where these parameters may be specified, not to describe these parameters in detail. This is done in subsequent sections. The MSC Nastran parameters may be specified on the command line, in a command initialization (INI) file, in runtime configuration (RC) files and, for some parameters, from environment variables. The information from these sources is consolidated at execution time into a single set of values. Much of this information is passed to analysis processing in a "**control file**", built using the templates (Customizing the Templates). (The records in this control file are echoed to the .log file.) Examples of INI and RC files are given in the User-Defined Keywords and Customizing Command Initialization and Runtime Configuration Files.

Command Initialization and Runtime Configuration Files

Although the purposes of the INI and RC files are somewhat different, the format of each file is the same. All INI and RC files are processed twice, once (the "first" pass) to extract parameters (keywords and other information) that are to be used for all MSC Nastran jobs, and once (the "second" pass) to extract parameters specific to a particular job. This is accomplished by separating the INI and RC files into a series of "sections" identified by a "section header" and "subsections" within sections, identified by a subsection "header." There are two types of sections: "unconditional" and "conditional." Subsections are always "conditional."

- An unconditional section is one that starts with the name of the section enclosed in square brackets ("`[",","]`"). Section names may not contain any embedded blanks but may be separated from the square brackets by any number of blanks. As currently implemented, there are three valid unconditional names: "`General`", "`Solver`" and "`Nastran`". (These section names are case-insensitive.) In addition, there is an implicit "unnamed" section that consists of all parameters in the INI or RC file that appear before the first named section or subsection. There is no special meaning assigned to any of the unconditional sections. Their use is optional; the section names are intended to be used for descriptive purposes.
- A conditional section or subsection is one that starts with an expression in the form:

  ```
  <keyword><operator><value>
  ```

  enclosed in section header identification characters. For a conditional section, the section header identification characters are square brackets ("`[",","]`"), just as for unconditional sections. For a subsection, the section header identification characters are "less than" and "greater than" ("`<",">`") characters. Keywords and values may not contain any embedded blanks but may be separated from each other and from the enclosing section header identification characters (the square brackets or "less than"-"greater than" characters) by any number of blanks. In the expression:

`<keyword>`	represents any valid internal keyword (see Keywords) or user-defined keyword (see User-Defined Keywords).
`<operator>`	specifies the comparison to be performed between `<keyword>` and `<value>` as follows: = equal (either string or numeric)

!	not equal (either string or numeric)
!=	not equal (either string or numeric)
<	numerically less than
<=	numerically less than or equal
>	numerically greater than
>=	numerically greater than or equal

`<value>` specifies the appropriate keyword value to be used in the comparison.

Keywords and values may be specified in any case.

Parameters in unconditional sections, but not in subsections (which are always conditional) within unconditional sections, are processed on the first pass through an INI or RC file. On the second pass, these parameters are ignored (they are not reprocessed). Parameters in conditional sections and subsections are ignored on the first pass. Parameters in conditional sections and subsections whose expressions evaluate to "true" are processed on the second pass through an INI or RC file, thus allowing conditional expressions to reference all of the valid keywords. Note that for subsections within conditional sections, both the conditional expression for the section *and* the conditional expression for the subsection must evaluate to "true" before parameters in the subsection are processed.

Parameter specifications in, either unconditional or conditional sections, may be continued, if necessary, by specifying a backslash (“\”) character as the last non-blank character of the line. Note for Windows users, if the parameter value itself ends with a backslash, the statement must have additional characters, such as a comment, after the value specification. For example, a specification such as:

```
sdir=e:\
```

will not work properly. Instead, write the statement as:

```
sdir=e:\ $ Specify the scratch directory
```

In addition to parameters, INI and RC files may contain “comment” records. There are two types of comment records: ignored and printed.

- Ignored comments are records that start with a semi-colon (";") or pound sign ("#"). These records are completely ignored. When running in Windows, there is a special form of ignored comments that may be specified in an INI file (but not in RC files). These are records that start with "REM", short for "REMARK". The test for "REM" is case-insensitive.
- Printed comments are records that start with the currency symbol ("$"). These records are passed on as part of the analysis information but are otherwise ignored.

Note:	Although sectioning within INI and RC files was first introduced in MSC Nastran 2004, valid INI and RC files from prior versions of MSC Nastran are fully compatible with this new format. Since sections were not supported in previous versions (except for INI files on Windows, which allowed unconditional sections), all parameters will be in the "unnamed" implicit section (or, on Windows, in named unconditional sections) and will be processed on the first pass through the file. No information will be extracted from these files on the second pass.

- Command Initialization (INI) File

  This file is used to define keywords that are to be set whenever the nastran command is executed. Typical keywords in the unconditional sections include the installation base directory and the version of MSC Nastran. Conditional sections and subsections might include keywords such as "rcmd" and "rsdirectory" in sections that are conditional upon the value of the "node" keyword.

  Default installation directories are:

  LINUX: *install_dir*/*prod_ver*/*arch*/nastran.ini

  Windows: *install_dir**prod_ver**arch*\nastran.ini

Starting with MSC Nastran 2011, there are two possible RC files that may be defined in each of the locations that are searched for RC files. The first name is a version independent name and the second name is a version dependent name, where the version number is indicated by <*vernum*> in the file name and the version number for MSC Nastran is 2023.1. The list below specifies the INI and RC files that MSC Nastran uses. Any or all of these files may be ommitted. Table 1-1 lists the keywords that are generally set in the unconditional sections of the command initialization file. Table 1-2 lists the keywords that are generally set in RC files.

prod_ver is the msc version (msc20231 for MSC Nastran 2023.1).

In addition, the default install_dir of MSC Nastran 2023.1 is as follows:

Linux: /msc/MSC_Nastran/2023.1

Windows: C:\Program Files\MSC.Software\MSC_Nastran\2023.1.

- System RC Files

  These files are used to define parameters that are applied to all MSC Nastran jobs using this installation structure. Many of the parameters that might be specified in the INI file could, alternatively, be specified in this file.

  LINUX: *install_dir*/conf/nastranrc and

  install_dir/conf/nast<*vernum*>rc

  Windows: *install_dir*\conf\nastran.rcf and

  install_dir\conf\nast<*vernum*>.rcf

- Architecture RC Files

  This files are used to define parameters that are applied to MSC Nastran jobs using this architecture.

  LINUX: *install_dir*/conf/*arch*/nastranrc and

  install_dir/conf/*arch*/nast<*vernum*>rc

  Windows: *install_dir*\conf*arch*\nastran.rcf and

  install_dir\conf*arch*\nast<*vernum*>.rcf

- Node RC Files

  These files are used to define parameters that are applied to MSC Nastran jobs running on this node. Alternatively, the parameters in this file could be specified in a conditional section in one of the previous files, using *nodename* as the value of the "s.hostname" keyword in the conditional expression.

  LINUX: *install_dir*/conf/net/*nodename*/nastranrc and

  install_dir/conf/net/*nodename*/nast<*vernum*>rc

  Windows: *install_dir*\conf\net*nodename*\nastran.rcf and

  install_dir\conf\net*nodename*\nast<*vernum*>.rcf

- User RC Files

  These files are used to define parameters that are applied to MSC Nastran jobs run by an individual user.

  LINUX: $HOME/.nastranrc and

  $HOME/.nast<*vernum*>rc

  Windows: %HOMEDRIVE%%HOMEPATH%\nastran.rcf and

  %HOMEDRIVE%%HOMEPATH%\nast<*vernum*>.rcf

- Local RC Files

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

  LINUX: .nastranrc and

  .nast<*vernum*>rc

  Windows: nastran.rcf and

  nast<*vernum*>.rcf

Please note that the LINUX 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.

Also, note that, on LINUX systems, the leading period (“.”) on the User RC Files and Local RC Files file names cannot be deleted even if alternate names are specified using the “a.urc” and “a.urcb” keywords as described below.

The file names listed above may be changed by the user using the “a.rc”, “a.rcb”, “a.urc” and “a.urcb” keywords, noting that the directories in which the files are located may not be changed.

- The “a.rc” keyword can be used to change the names of the version dependent RC file names for the System RC Files, the Architecture RC Files and the Note RC File. The default for this keyword is “nast<*vernum*>rc” for LINUX and “nast<*vernum*>.rcf” for Windows.
- The “a.rcb” keyword can be used to change the names of the version-independent RC file names for the System RC Files, the Architecture RC Files and the Node RC Files. The default for this keyword is “nastranrc” for LINUX and “nastran.rcf” for Windows.
- The “a.urc” keyword can be used to change the names of the version dependent RC file names for the User RC Files and the Local RC Files. For LINUX, the default for this keyword is the value of the “a.rc” keyword with a leading period (“.”) added. For Windows, the default for this keyword is the value of the “a.rc” keyword.
- The “a.urcb” keyword can be used to change the names of the version-independent RC file names for the User RC Files and the Local RC Files. For LINUX, the default for this keyword is the value of the “a.rcb” keyword with the leading period (“.”) added. For Windows, the default for this keyword is the value of the “a.rcb” keyword.

In addition to keyword specifications, the following MSC Nastran statements (from the NASTRAN and FMS Sections) may appear in RC files and conditional sections in an INI file: NASTRAN, ACQUIRE, ASSIGN, CONNECT, DBCLEAN, DBDICT, DBFIX, DBLOAD, DBLOCATE, DBSETDEL, DBUNLOAD, DBUPDATE, DEFINE, ECHOOFF, ECHOON, ENDJOB, EXPAND, INCLUDE, INIT, PROJ, RESTART and RFINCLUDE. Except for minimal checking of the NASTRAN and PARAM statements, the syntax of these statements is not validated. These records are simply passed on for use in MSC Nastran analysis processing.

INI files and RC files also may contain PARAM statements that specify values that affect MSC Nastran analysis processing. The values associated with PARAM names may be specified using PARAM statements in INI files and RC files or by using PARAM keywords, defined using the PARAM keywords feature as described in User-Defined Keywords, 143. PARAM statements must be specified in "free-field format", i.e., in the Case Control PARAM format (PARAM,name,value), not in Bulk Data fixed-field format. Please see Parameters (Ch. 6) in the *MSC Nastran Quick Reference Guide* for more information on PARAM names and statements and their usage.

Environment Variables

Several keywords may have their values set from associated environment variables. When this is the case, the environment variable takes precedence over any INI or RC file keyword specification. A command-line specification will over-ride the environment variable specified value. This same precedence rule applies to user-defined keywords that may have their initial values taken from environment variables, as described in the next section. A list of the keywords and their associated environment variables, along with a description of each keyword, may be obtained by using the following command:

nast_ver help env

User-Defined Keywords

In addition to the internally defined keywords (see Keywords, 172), MSC Nastran allows users to define their own keywords. There are two classes of user-defined keywords:

- General keywords. These are intended for use in INI file or RC file conditional section clauses, in user modifications to the run template files (nastran.dmp, nastran.lcl, nastran.rmt or nastran.srv) and, for LINUX, in customized queue commands ("submit" keyword).
- PARAM keywords. These are keywords associated with a PARAM name. Using descriptive keywords to set a PARAM value may be more convenient than specifying the PARAM statement in an RC file. Also, keywords are not limited to a maximum of eight characters, as PARAM names are, and may be more descriptive of the action being affected or requested.

User-defined keywords are supported by the "help" and "whence" functions.

General Keywords

These keywords are defined in the file specified by the "0.kwds" keyword. While the file is not delivered, an administrator could create the file. The default file names are:

LINUX:	*install_dir*/***prod_ver***/*arch*/nastran.kwds
Windows:	*install_dir****prod_ver****arch*\\nastran.kwds or *install_dir*\\bin\\nast20231.kwds The file used is the *first* one found.

The records in this file consist of:

- Comment records. These are records that start with a comment character (hash, '#', semi-colon, ';', or currency symbol, '$') and are completely ignored.
- Blank or null records. These records are ignored.
- Keyword records. These records consist of the keyword name along with an optional value descriptor and comment in the form:

  ```
  keyword_name[,attributes] : value_descriptor comment
  ```

  where:

`keyword_name`	is the name to be assigned to the user keyword. This name may not contain any embedded blanks and may not be the same as any internal keyword or previously specified user-defined keyword. It is also case-insensitive except in the case when its initial value may be set from an environment variable with the same name.
`attributes`	specifies optional attributes to be assigned to the keyword defined by keyword-name. Currently, the only defined attribute is: `argv` keyword and its value is to be added to the “r.argv” keyword value Any number of blanks may separate `keyword_name`, the separating command and the attributes specification.
`value_descriptor`	is optional. If specified, it should be as described in Value Descriptors, 145 and may not contain any embedded blanks. If this field is not present, the separating colon may be omitted.. The default value descriptor is `"string"`. This field may also specify that the initial value of this keyword be taken from an environment variable with the same name.
`comment`	is an optional comment field. If present, it must be separated from `value_descriptor or keyword_name` by blanks or must begin with a comment character.

There may be any number of leading blanks in the record and before and after the separating colon.

General keywords and the values assigned to them only affect MSC Nastran processing if:

- there are customized INI and RC files that have conditional sections, using these keywords in expressions, that specify other keywords and statements (e.g., NASTRAN and PARAM statements) that modify MSC Nastran processing to meet the requirements of a user's site and installation.
- they are used in customized templates (Customizing the Templates).
- for LINUX systems, they are used in customized queue commands defined using the "submit" keyword (Customizing Queue Commands (LINUX)).

PARAM Keywords

These keywords are defined in the file specified by the "0.params" keyword The default file names are:

LINUX:	*install_dir*/***prod_ver***/*arch*/nastran.params
Windows:	*install_dir****prod_ver****arch*\nastran.params

The records in this file consist of:

- Comment records. These are records that start with a comment character (hash, '#', semi-colon, ';', or currency symbol, '$') and are completely ignored.
- Blank or null records. These records are ignored.
- Keyword-name records. These records consist of the keyword name, the associated PARAM name, along with an optional value descriptor and comment in the form:

  keyword_name : param_name : value_descriptor comment

  where:

`keyword_name`	is the name to be assigned to the PARAM keyword. This name is case-insensitive, may not contain any embedded blanks and may not be the same as any internal keyword, general user-defined keyword or previously specified PARAM keyword.
`param_name`	is the PARAM name to be associated with `keyword_name`. This name is case-insensitive, may be a maximum of eight characters, must begin with an alphabetic character and may not contain any embedded blanks. Also, it may not be the same as any previously specified PARAM name.
`value_descriptor`	is optional. If specified, it should be as described in Value Descriptors and may not contain any embedded blanks. If this field is not present, the separating colon may be omitted. The default value descriptor is `"string"`.
`comment`	is an optional comment field. If present, it must be separated from `value_descriptor` or `param_name` by blanks or must begin with a comment character.

There may be any number of leading blanks in the record and before and after the separating colons.

Keyword names that are the same as PARAM names are allowed, as long as the keyword name is not an internal or general user-defined keyword name.

Values associated with PARAM names, whether set using PARAM keywords or set using PARAM statements (statements having the form PARAM,*name*,*value*), directly affect MSC Nastran analysis processing.

Value Descriptors

Value descriptors enable limited syntax checking for values assigned to general and PARAM user-defined keywords. For general keywords, they may also specify that the initial value of the keyword be set from the value associated with the environment variable having the same name as the keyword. There are two types of syntax checking available: value must be one of a list of entries or value must be numeric. Also, the two forms can be combined. These are specified as follows:

List: {"*val1*","*val2*",...,"*valn*"}

That is, the acceptable values are enclosed in double quotes (") and separated from each other by commas. The specification, including the various acceptable values, may not contain any embedded blanks. Values are case-insensitive and any partial specification is acceptable and will be replaced by the full value. For example, if a keyword may only have the values "preliminary", "check" and "final", the value descriptor would be:

```
{"Preliminary","Check","final"}
```

and a value specification of "`Ch`" would be accepted and replaced by "`check`".

Numeric: `number`

Values will be checked to see if they are valid numbers, either integer or floating point. For example, valid keyword value specifications could be: "`1`", "`-3.247`", "`4.e-5`". "`3.75-4`", "`4.24x`" and "`-4-5`" are invalid specifications.

Note:	This checking does *not* support the NASTRAN "`nnnseee`" numeric format, where the 'e' between the number and the signed exponent ("`seee`") is missing.

Complex value: `number,number`

This format is only supported for PARAM keyword value descriptors. Values will be checked to see if they consist of two valid numeric values, separated by a comma.

Combined: `{"val1","val2",...,"valn",number}`

Note:	This "combined" format does not support complex numbers.

In addition, for general keywords, if the value descriptor starts or ends with the string "`env`", specified in any case and separated from the rest of the value descriptor with a comma (unless the value descriptor is only "`env`"), the keyword value will be set using the value associated with the environment variable having the same name as the keyword. The environment value will be subjected to the same syntax-checking rules that an INI file, RC file or command line specification would be, with a warning message generated if syntax checking fails. This occurs even if the keyword is specified on the command line. Note that, for LINUX systems, since environment variable names are case-sensitive, the keyword name must be specified exactly the same as the environment variable name. This is the only time that the keyword name is case-sensitive. For Windows systems, since environment variable names are not case-sensitive, this restriction does not apply. Keyword values set from environment variables over-ride keyword values set in INI or RC files but do not over-ride keyword values set on the command line.

If a value descriptor is omitted or is not one of these formats, no syntax checking will be performed.

Examples:

1. The following value descriptor would accept a value of "test", "final" or a number:

```
{"Test","Final",Number}
```

Acceptable values would be: `te` (replaced by test), `FIN` (replaced by final), `7, 14.5, 3.e-4, -5`

2. The following value descriptor would accept only the strings "abc", "def", "ghi" and "glm":

```
{"abc","def","ghi","glm"}
```

Acceptable values would be: `g` (replaced by `ghi`), `aB` (replaced by `abc`), `gl` (replaced by `glm`), `D` (replaced by `def`)

3. The following value descriptor, only valid for a PARAM keyword, would only accept a complex number specification:

```
number,number
```

Acceptable values would be: `1,2, 7.54,3.14`

4. The following value descriptors, only valid for a general keyword, would accept only the strings "qrs", "test", and "xyz". In addition, the value descriptor requests that the keyword value be set from the environment.

```
enV,{"qrs","test","xyz"}
```

or

```
{"qrs","test","xyz"},Env
```

Acceptable values would be: `q` (replaced by qrs), `xY` (replaced by xyz), `T` (replaced by test)

Resolving Duplicate Parameter Specifications

MSC Nastran processing information is obtained by scanning the various INI and RC files, the system environment, and the Nastran command line in the following order:

1. Nastran command line, first pass. Only "program options", i.e., "*-x*" options, are processed during this command line scan. For example, this is when the "-i *ini_file_name*" program option is processed.
2. Environment variables, first pass. During this pass, the only keywords whose values are set are those that may only be specified as environment variables. This includes keywords such as HOME (for LINUX), HOMEDRIVE and HOMEPATH (for WINDOWS) and PWD.
3. INI file, first pass, if this file exists. During this pass, only unconditional sections are processed. Generally, the only keywords processed in this pass are: 0.kwds, 0.params, accmd, acvalid, rcmd, rsdirectory, sysmsg and version (although rcmd and rsdirectory probably should be in conditional sections scanned during the second pass).
4. Environment variables, second pass. During this pass, only those keywords that may only be set in global sections of the INI file or as environment variables are processed. This includes keywords such as MSC_ARCH, MSC_BASE and MSC_VERSD.
5. Nastran command line, second pass. The only general use keywords processed during this command line scan are: dmparallel, jid, jidpath, jidtype, node, pause, rcf, username, version and whence. The processing of other command line keywords is deferred until later command line scans.

This is the time that the user-defined keyword definition files (for both general use and PARAM keywords), if any, are processed and the keyword specifications defined by these files are added to the keywords tables. The keywords defined in these files may be used just as internal keywords are used. (See User-Defined Keywords.)

6. System RC files, first pass, if these files exist. During this pass, only unconditional sections are processed.
7. Architecture RC files, first pass, if these files exist. During this pass, only unconditional sections are processed.
8. Node RC files, first pass, if these files exist. During this pass, only unconditional sections are processed.
9. User RC files, first pass, if these files exist. During this pass, only unconditional sections are processed.
10. Local RC files, first pass, if these files exist. During this pass, only unconditional sections are processed.
11. Environment variables, third pass. During this pass, only "general" user-defined keywords that have been flagged to be set from environment variables are processed. (This pass will be skipped if there are no "general" user-defined keywords.)
12. Nastran command line, third pass. Only "general" user-defined keywords are processed during this command line scan. (This pass will be skipped if there are no "general" user-defined keywords.)

At this point, all keyword values that can be used in conditional section expressions are known.

13. INI file, second pass, if this file exists and has conditional sections. During this pass, only the conditional sections are processed.
14. System RC files, second pass, if these files exist and have conditional sections. During this pass, only the conditional sections are processed.
15. Architecture RC files, second pass, if these files exist and have conditional sections. During this pass, only the conditional sections are processed.
16. Node RC files, second pass, if these files exist and have conditional sections. During this pass, only the conditional sections are processed.
17. User RC files, second pass, if these files exists and have conditional sections. During this pass, only the conditional sections are processed.
18. Local RC files, second pass, if these files exist and have conditional sections and if they are not ignored. During this pass, only the conditional sections are processed.
19. Environment variables, fourth pass. During this pass, all keywords that may be set from environment variables and that have not been processed previously are now processed.
20. Nastran command line, fourth pass. All keywords not processed during the previous passes are now processed. For example, this is when user-defined PARAM keyword specifications are processed.

    At this point, all information necessary to generate the "control file" has been collected. This file is generated when the "script templates" (see Customizing the Templates) are processed.
21. NASTRAN, FMS and PARAM statements in the input file.

If duplicate keywords are encountered, the *last* specification found is the one used. That is, the above list specifies the precedence order, from lowest precedence (number 1) to highest (number 21). The only case in which the last keyword specification is not used is when keywords are "locked", i.e., when a specification of the form

```
lock=keyword
```

is processed. After this "lock" request is processed, any requests to set `keyword`, whether from INI files, RC files, environment variables or command line arguments, are quietly ignored. That is, processing proceeds as if any `keyword` specifications specified after the "`lock=keyword`" request do not exist. Once a keyword has been "locked," there is no way to "unlock" it. (Note that it is valid to "lock" the `lock` keyword itself.)

If duplicate NASTRAN and FMS statements are encountered, they are simply passed on for use in MSC Nastran analysis processing in the order in which they were encountered.

Thus, the general rule for resolution is:

- Information specified in NASTRAN input data files always takes precedence over any other values.
- Command line parameters have the next highest precedence.
- Environment variables associated with keywords and that have non-null values are next.
- RC file parameter specifications are next.
- INI file parameter specifications are last.

Generally, the only exceptions to this precedence ordering are "general" user-defined keyword specifications. The command line values take precedence over values specified in unconditional INI file and RC file sections but have lower precedence than values specified in conditional INI file and RC file sections. Because the primary purpose for general user-defined keywords is for conditional section selection, changing a general user-defined keyword in a conditional section *may* lead to unexpected results. Such specifications should be used with care. Also, because user-defined PARAM keywords on the command line are not processed until the last command line scan, PARAM keywords should not be used in INI file and RC file conditional section expressions since command line specified values will not be in effect when these expressions are evaluated.

Because PARAM values may be specified either using PARAM statements or using PARAM keywords, they require further explanation. PARAM statements and PARAM keywords referring to the same PARAM name are considered equivalent definitions for the PARAM name. As such, the last specification, regardless of whether it was a PARAM statement or a PARAM keyword, is the one that is used to establish the value associated with the PARAM name.

Customizing Command Initialization and Runtime Configuration Files

Table 1-1 lists the keywords that are generally set in the unconditional sections of the command initialization file.

Table 1-1 Command Initialization File Keywords

Keyword	Purpose
0.kwds	Alternate name for user-defined keywords definition file.
0.params	Alternate name for PARAM keywords definition file
acct	Enables job accounting, see Enabling Account ID and Accounting Data.
acvalid	Activates account ID validation, see Enabling Account ID Validation.
MSC_BASE	Defines the installation base directory.
version	Specifies the default version of MSC Nastran to be run.

Most of the command line keywords can be set in any of the RC files. Table 1-2 lists keywords that are generally set in the system, architecture, or node RC files:

Table 1-2 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.
buffsize	System	Set the default buffsize. Suggested values are in Table 4-5.
lock	Any	Prevent further changes to a keyword's value.
memory	Node	Specifies a default memory allocation
memorymaximum	Node	Specifies a maximum "memory" request. May be specified as a percentage of RAM e.g. `memorymax=0.5xPhysical`
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 file.

Table 1-2 RC File Keywords (continued)

Keyword	Preferred RC File	Purpose
post	Architecture	LINUX: Specifies commands to be run after each job is completed.
ppcdelta	Architecture	LINUX: Specifies the value that is subtracted from the "CPU" keyword value to determine the NQS per-process CPU time limit.
ppmdelta	Architecture	LINUX: Specifies the value that is added to the "memory" keyword value to determine the NQS per-process memory limit.
pre	Architecture	LINUX: Specifies commands to be run before each job begins.
prmdelta	Architecture	LINUX: Specifies the value that is added to the "ppm" value to determine the NQS per-request (per-job) memory limit.
qoption	Architecture	LINUX: Specifies a string of additional queuing options to be set in the queue submittal command.
rcmd	Any	Specifies the remote Nastran command to be used when "node" is specified. Should be in a conditional section using "node" in the conditional expression.
real	Node	Specifies the "REAL" parameter to limit virtual memory usage.
rsdirectory	Any	Specifies the scratch directory to be used when "node" is specified. Should be in a conditional section using "node" in the conditional expression.
scratch	Any	Specifies the default job status as scratch or permanent.
sdirectory	Node	Specifies a default scratch directory.
submit	Architecture	LINUX: Defines queues and their associated submittal commands.
sys*n*	Any	Specifies system cells. Can also be specified using the synonym keywords, e.g., buffsize is equivalent to sys1.

Examples

The following (relatively simplistic) examples illustrate how unconditional and conditional sections could be used.

Example 1:

Assumptions: There are three computer nodes, sysnode1, sysnode2 and sysnode3, that may be accessed.

On sysnode1:

- MSC Nastran 2014 and MSC Nastran 2023.1 are installed:
  - MSC Nastran 2014 is accessed using "/local/msc/bin/nast2014"
  - MSC Nastran 2023.1 is accessed using "/local/msc/bin/nast20231"
  - The scratch directory is /local/temp

On sysnode2:

- Only MSC Nastran 2014 is installed and is accessed using "/local1/msc/bin/nast2014"
- The scratch directory is /local1/temp

On sysnode3:

- MSC Nastran 2014 and MSC Nastran 2023.1 are installed:
  - MSC Nastran 2014 is accessed using "/local2/msc/bin/nast2014"
  - MSC Nastran 2023.1 is accessed using "/local2/msc/bin/nast20231"
- The scratch directory is /local2/temp

All of this information could be specified in an INI file, identical on all three nodes, as follows:

```
;
; This is the MSC Nastran Command Initialization File
; The default version is to be set to 2023.1
;
version=2023.1
; Define conditional sections giving the appropriate sdir
; values when MSC Nastran is run locally.

[ s.hostname  =  sysnode1 ]
sdir=/local/temp
[ s.hostname  =  sysnode2 ]
sdir=/local1/temp
[ s.hostname = sysnode3 ]
sdir=/local2/temp

; Define conditional sections giving the appropriate
; remote access keywords when a "node" value,
; requesting remote execution, is specified.
;
[ node = sysnode1 ]
rsdir=/local/temp
< version = 2014.0 >
rcmd=/local/msc/bin/nast2014
```

```
< version = 2023.1 >
rcmd=/local/msc/bin/nast20231

[ node = sysnode2 ]
rsdir=/local1/temp
< version = 2014.0 >
rcmd=/local1/msc/bin/nast20140

[ node = sysnode3 ]
rsdir=/local2/temp
< version = 2014.0 >
rcmd = /local2/msc/bin/nast2014
< version = 2023.1 >
rcmd=/local2/msc/bin/nast20231

;
; This is the end of the Command Initialization file
;
```

Alternatively, the information could be split between an INI file and a system RC file, identical on all three nodes, as follows:

In the INI file:

```
;
; This is the MSC Nastran Command Initialization File
; The default version is to be set to 2023.1
;
version=2023.1

; Define conditional sections giving the appropriate
; remote access keywords when a "node" value,
; requesting remote execution, is specified.
;
[ node = sysnode1 ]
rsdir=/local/temp
< version = 2014.0 >
rcmd=/local/msc/bin/nast2014
< version = 2023.1 >
rcmd=/local/msc/bin/nast20231

[ node = sysnode2 ]
rsdir=/local1/temp
< version = 2014.0 >
rcmd=/local1/msc/bin/nast20140

[ node = sysnode3 ]
rsdir=/local2/temp
< version = 2014.0 >
rcmd = /local2/msc/bin/nast2014
< version = 2023.1 >
rcmd=/local2/msc/bin/nast20231

;
; This is the end of the Command Initialization file;
```

In the system RC file, identical on all three nodes:

```
;
; This is the MSC Nastran system RC file.
;
```

```
; Define conditional sections giving the appropriate sdir
; values when MSC Nastran is run locally.

[ s.hostname  =  sysnode1 ]
sdir=/local/temp
[ s.hostname  =  sysnode2 ]
sdir=/local1/temp
[ s.hostname = sysnode3 ]
sdir=/local2/temp

;
; This is the end of the system RC file
;
```

Example 2:

Assumptions: User keywords defining "run type" and "data complexity" are needed and AUTOSPC, AUTOSPCR, BAILOUT and ERROR PARAM values are to be set based on these keywords.

The nastran.kwds file could be:

```
; User Keywords
Runtype:{"prelim","development","final"};Analysis stage
     Level :    number       # Data complexity level
     ;
```

The nastran.params file could be:

```
; PARAM keywords

Set_AutoSPC : AutoSPC   : {"Yes","No"}
Set_AutoSP_CR    : AUTOSPCR : {"yes","no"}
Bailout_Value : bailout : number
Set_Error : Error : number

;
```

Then, the system RC file could contain:

```
; RC file
[ runtype = prelim ]
set_autospc = yes
bailout_value = -1
set_error = 0
set_autosp_cr = yes

[ runtype = development ]
set_autospc=yes
bailout_value=0
set_error=-1

[runtype=final]
set_autospc=no
param,bailout,0
param,error,-1
param,autospcr,no

[level < 3]
; basic data complexity parameters
[level >= 3]
<level>8>
```

```
; advanced data complexity parameters

<level<=8>
; intermediate data complexity parameters

; End of RC file
```

Symbolic Substitution

Introduction

Symbolic Substitution is a capability added to MSC Nastran that allows a user to effectively modify a Nastran data file using command line and RC file keyword specifications without actually editing the file. This capability is very similar to "environment variable" expansion that happens in various command prompt shells such as the Linux Bourne, Korn and C shells and the Windows Command Prompt shell when scripts are processed. It is also analogous in some ways to the capabilities provided by programming language preprocessors, for example, the CPP preprocessor used by the various C/C++ compilers. The key feature of symbolic substitution is that these modifications do *not* affect the actual data file but present the data read from the data file to the processing program as if it was the modified data that was being processed.

Generally, symbolic substitution means that a data record is scanned to see if it contains special data strings (that identify the "symbolic" variables) that specify "symbolic substitution" requests. If such strings are found, the record is modified to replace the special data strings with user-defined substitution (replacement) strings (the values currently associated with the "symbolic" variables, i.e., the variable "values") and it is this modified record that is actually processed. This symbolic substitution happens before any other processing of the record occurs, thus making it transparent to the rest of the program processing the data record. In the case of MSC Nastran, this symbolic substitution processing will happen immediately after a record is read from the Nastran data file and before any other processing (with the possible exception of special processing required to satisfy licensing requirements) is performed.

Simple Examples

Two very simple examples illustrate how this capability could be used in Nastran data files. Note that the details of the syntax are completely described in the following sections and may be ignored for now. Also note that the examples do not deal with things such as managing the output from multiple Nastran runs. These issues, involving, among other techniques, using command line or RC file keywords such as "`out=`", "`append=`" and "`old=yes`", are beyond the scope of this document.

Example 1:

Suppose you want to make several tests where the thickness of a PSHELL element is to be varied. You could do this by defining the thickness of the PSHELL element as a "symbolic variable" (identified using the string "`%thickness%`"), setting a default value (using the "`%defrepsym`" statement) and specifying the desired thickness on the command line (using the "`REPSYM=`" keyword). A very simple data file (`sym.dat`) could be (where most of the BULK entries are in an include file named "`model.bdf`", not shown here):

```
%defrepsym thickness=5.0
SOL 103
CEND
TITLE = 1st perturbation, t = %thickness%
ECHO = NONE
SUBCASE 1
   METHOD = 100
   SPC = 1
```

```
   DISP = ALL
BEGIN BULK
EIGRL,100,,,6
PARAM,POST,0
PARAM,GRDPNT,0
$PBEAML Properties
PBEAML   2       1               I
         70.0    60.0    60.0    3.3     5.      5.
$
$PSHELL Properties
$
pshell,1,1,%thickness%,1,,1
$
include 'model.bdf'
enddata
```

If the test is run using the following command line:

```
nast20231 sym repsym=thickness=1.0 ...
```

the test will run as if the "TITLE" and "pshell" records are:

```
TITLE = 1st perturbation, t = 1.0
```

and

```
pshell,1,1,1.0,1,,1
```

If the test is run using the following command line:

```
nast20231 sym repsym=thickness=3.5 ...
```

the test will run as if the "TITLE" and "pshell" records are:

```
TITLE = 1st perturbation, t = 3.5
```

and

```
pshell,1,1,3.5,1,,1
```

If the test is run without specifying any REPSYM setting for "thickness", e.g., using the following command line:

```
nast20231 sym ...
```

the test will run as if the "TITLE" and "pshell" records are:

```
TITLE = 1st perturbation, t = 5.0
```

and

```
pshell,1,1,5.0,1,,1
```

Example 2:

Suppose you have a test that contains two superelements, where the only difference between the data for each superelement is the area of a PBAR element. Instead of having two different definitions, you could have a single definition of the data in an include file, where the area of the PBAR is specified as a symbolic variable. The include file (called "bar.bdf") could be:

```
%defrepsym area=1.
grid,2,,1.0,0.0,0.0
grid,3,,2.0,0.0,0.0
```

```
grid,4,,3.0,0.0,0.0,,123456
cbar,2,2,2,3,0.,1.,0.
cbar,3,2,3,4,0.,1.,0.
pbar,2,2,%area%,1.,1.,1.
mat1,2,1.e7,,.3
```

and the actual input file could be:

```
sol 101
cend
title=simple part se
echo=both
subcase 1
load=1
disp=all
elforce=all
begin bulk
grid,1,,0.0,0.0,0.0
grid,2,,1.0,0.0,0.0
cbar,1,1,1,2,0.,1.,0.
pbar,1,1,1.,1.,1.,1.
mat1,1,1.e7,,.3
force,1,1,,1.,1.,1.,1.
$
begin super=1
%setrepsym area=1.
include 'bar.bdf'
$
begin super=2
%setrepsym area=2.
include 'bar.bdf'
enddata
```

The first "`include 'bar.bdf'`" statement will be processed as if the `pbar` record is

```
pbar,2,2,1.,1.,1.,1.
```

and the second "`include 'bar.bdf'`" statement will be processed as if the `pbar` record is

```
pbar,2,2,2.,1.,1.,1.
```

Detailed Specifications

The use of the Symbolic Substitution capability is defined by a number of "rules". These "rules" are given in the following sections and provide the complete specification. Following the rules, there is information about requesting report information and about error handling. Finally, there are some (again simple) examples showing usage.

Symbolic Substitution Rules

The following rules define the symbolic substitution user interface. The descriptions start with the rules for variable naming, followed by the rules for defining the replacement width information, followed by the various keywords and statements used to control symbolic substitution.

Variable Naming

The rules for naming symbolic substitution variables are:

- Symbolic variable names are not case-sensitive, are a maximum of 32 characters long and may not contain leading, trailing or embedded blanks or special characters including (“_”). Variable names must start with an alphabetic character followed by zero or more alphabetic or numeric characters. For example:
  - The variable name "VaRiaBLe1" is the same as "VARIABLE1" and "variable1"
  - The following variable names are valid:

- abcdef
- abc123
- Name1

  - The following variable names are not valid:

- 123abc Does not start with an alphabetic character
- a bcd Contains an embedded blank
- abc& Contains an invalid character ('&')
- /def Does not start with an alphabetic character
- _abc123 Uses an underscore in the name.

- Unless symbolic variable values are quoted, they are not case-sensitive and may not contain leading, trailing or embedded blanks or percent ('%') characters. The quoting rules are given below.

Substitution Field Width Specification

The ability to control the appearance of any symbolic substitution is an important requirement when generating data for a program such as MSC Nastran. The result of a symbolic substitution request is identified as a *field*. Substitution field width information can be taken by default, specified in the data file or specified using command line and/or RC file keywords. These methods are explained below.

The rules for defining substitution field width information are:

- Symbolic variable substitution is, by default, *exact*. That is, the number of characters occupied by the symbolic symbol replacement is exactly the same as the replacement value. However, this default replacement processing can be controlled by specifying the substituted field *width*, the field *precision* and the *justification* within the field. This information is specified using the syntax

  `-w.p`

  where the ‘`-`‘, ‘`w`’ and ‘`p`’ are all optional and have the following meanings.
  - The field width specification (`w`) defines the *minimum* number of characters the field is to have as a decimal integer value. If the replacement value has fewer characters than the field width, it will be padded with spaces on the left (by default) or on the right (if the left justification flag is specified). If the replacement value has more characters than the field width and if no precision

value was specified, the entire replacement string will be used. A field width value of 0 (zero) is equivalent to omitting the width specification. Note that a negative width value will be processed as if the "left-justification" flag was specified (see below) since a negative field width is meaningless.

- The field precision specification (`p`) defines the *maximum* number of characters the field is to have. The format is a period (`.`) followed by a decimal integer value. If the replacement value length exceeds the precision value, only the last `p` (by default) or the first `p` (if the left justification flag is set) characters of the replacement value will be used. A field precision value of 0 (zero) (or a negative value) is equivalent to omitting the precision specification.
- If both field width and field precision are specified and are positive, the precision value cannot be less than the width value. If it is, it will be reset to the field width.
- The '`-`' character is the "left-justification" flag and specifies that the replacement value is to be left-justified within the field. If this character is omitted, the replacement value will be right-justified within the field.

- For example, the width, precision and justification of a typical field in the Bulk Data portion of a Nastran data file is:

  `-8.8`

  meaning that the field is exactly eight characters wide and that data is to be left-justified within the field. For a wide-format Bulk Data record, this specification would be:

  `-16.16`

  The specification for an exact replacement, i.e., where the replaced field is exactly the size of the replacement value, is:

  `0.0`
- To simplify width specification for Nastran widths, the following (case-insensitive) synonyms for common widths are available and may be used wherever a width specification can be used:

`exact`	is equivalent to `0.0`
`bulk`	is equivalent to `-8.8`
`wide`	is equivalent to `-16.16`

It is very important to note that there are two distinct portions to a Nastran data file, that part that is before the first `BEGIN` statement and that has "free format", and that part that is after the first `BEGIN` statement (the Bulk Data Section) and often has fixed format fields. Because of this, two different sets of field width information are maintained for use when field width information is not explicitly specified as part of a symbolic substitution request, one for use before the first `BEGIN` statement and one for use after the first `BEGIN` statement.

Defining Variable Values and Width Information

Symbol names and associated values and symbol width specifications may be set using keywords on the command line or in RC files and may be set using special statements in the Nastran data file itself. Each keyword and statement is explained in detail.

Using Command Line or RC File Keywords

Setting Variable Value Using REPSYM

Symbolic variables and associated values may be set on the Nastran command line or in RC files using the keyword

```
repsym=<varname>=<varvalue>
```

where <varname> specifies the name of the symbolic variable and <varvalue> specifies the value to be associated with the variable name. For example,

```
repsym=abc=1.23e-5
```

Setting Variable Width Information Using REPWIDTH

Symbolic variable substitution default width information may be set on the Nastran command line or in RC files using the keyword

```
repwidth=<widthinfo1>,<widthinfo2>
```

where <widthinfo1> specifies the default width information for the portion of the Nastran data file before the BEGIN statement and <widthinfo2> specifies the default width information for the portion of the Nastran data file after the BEGIN statement. Each is specified using a -w.p specification or as one of the synonyms, as described previously. If either width specification is omitted, the current default for that section is not changed. Note that the separating comma is required if the Bulk Data Section width value is to be set, i.e., if <witdhinfo2> is specified. For example,

```
repwidth=12,bulk
```

specifies that symbolic substitution default width is to be 12.0 before the BEGIN statement is encountered and -8.8 after the BEGIN statement is encountered and

```
repwidth=,bulk
```

specifies that symbolic substitution default width is to be EXACT (or 0.0, the default) before the BEGIN statement is encountered and -8.8 after the BEGIN statement is encountered.

Just as with other Nastran command line or RC file keywords, the REPSYM and REPWIDTH keywords are not case-sensitive.

Using Special Statements in a Nastran Data File

Setting Values Using setrepsym

Symbolic variables and associated values may be set in a Nastran data file using the following statement:

```
%setrepsym <varname>=<varvalue>
```

where the '%' character *must* be in column 1 and nothing else may appear in the record except for optional comments following <varvalue>, where the start of the comment is indicated by a ' $' (blank, currency symbol). The setrepsym string is not case-sensitive and at least one blank must separate this string from the <varname> specification. For example,

```
%setrepsym abc=1.23e-5
```

Clearing ("Unsetting") Values Using unsetrepsym

A symbolic variable value set using the %setrepsym statement may cleared ("unset") in a Nastran data file using the following statement:

```
%unsetrepsym <varname>
```

where the '%' character *must* be in column 1 and nothing else may appear in the record except for optional comments following <varname>, where the start of the comment is indicated by a ' $'. The unsetrepsym string is not case-sensitive and at least one blank must separate this string from the <varname> specification. For example, to clear the variable abc, use

```
%unsetrepsym abc
```

Setting Default Values Using defrepsym

Default variable values can be set in a Nastran data file using the following statement:

```
%defrepsym <varname>=<varvalue>
```

where the '%' character *must* be in column 1 and nothing else may appear in the record except for optional comments following <varvalue>, where the start of the comment is indicated by a ' $'. The defrepsym string is not case-sensitive and at least one blank must separate this string from the <varname> specification. The specified value will be used *only* if a value for <varname> was not previously set, i.e., by a repsym keyword on the command line or in an RC file or by a %setrepsym statement previously specified in the data file that has not been unset by a %unsetrepsym statement. For example,

```
%defrepsym abc=2.46e+2
```

Clearing ("Unsetting") Default Values Using undefrepsym

The default value for a symbolic variable may cleared ("unset") in a Nastran data file using the following statement:

```
%undefrepsym <varname>
```

where the '%' character *must* be in column 1 and nothing else may appear in the record except for optional comments following <varname>, where the start of the comment is indicated by a ' $'. The undefrepsym string is not case-sensitive and at least one blank must separate this string from the <varname> specification. For example, to clear the default value associated with variable abc, use

```
%undefrepsym abc
```

Setting Width Information Using setrepwidth

Symbolic variable substitution default width information may be set in a Nastran data file using the following statement:

```
%setrepwidth <widthinfo1>,<widthinfo2>
```

where the '%' character *must* be in column 1 and nothing else may appear in the record except for optional comments following <widthinfo2>, where the start of the comment is indicated by a ' $'. The setrepwidth string is not case-sensitive and at least one blank must separate this string from the width specifications. There may not be any blanks within the width specifications. <widthinfo1> specifies the width information for the portion of the Nastran data file before the BEGIN statement and <widthinfo2> specifies the width information for the portion of the Nastran data file after the BEGIN statement. Each is

specified using a -w.p specification or as one of the synonyms, as described above. If either width specification is omitted, the current width information for that section is not changed. Note that the separating comma is required if the Bulk Data Section width value is to be set, i.e., if <widthinfo2> is specified. For example,

```
%setrepwidth 0.0,wide
```

specifies that the symbolic substitution width specification is to be 0.0 before the BEGIN statement and is to be -16.16 after the BEGIN statement.

Clearing ("Unsetting") Width Information Using unsetrepwidth

Symbolic variable substitution width information set using the %setrepwidth statement may be cleared in a Nastran data file using the following statement:

```
%unsetrepwidth
```

where the '%' character *must* be in column 1 and nothing else may appear in the record except for optional comments following the unsetrepwidth string, where the start of the comment is indicated by a ' $'. The unsetrepwidth string is not case-sensitive and must be followed by at least one blank. This statement does not have any arguments and clears both width specifications.

Setting Default Width Information Using defrepwidth

Default symbolic variable substitution width information may be set in a Nastran data file using the following statement:

```
%defrepwidth <widthinfo1>,<widthinfo2>
```

where the '%' character *must* be in column 1 and nothing else may appear in the record (except for optional comments following <widthinfo2>, where the start of the comment is indicated by a ' $'. The defrepwidth string is not case-sensitive and at least one blank must separate this string from the width specifications. There may not be any blanks within the width specifications. <widthinfo1> specifies the default width information for the portion of the Nastran data file before the BEGIN statement and <widthinfo2> specifies the default width information for the portion of the Nastran data file after the BEGIN statement. Each is specified using a -w.p specification or as one of the synonyms, as described above. If either width specification is omitted, the current width information for that section is not changed. Note that the separating comma is required if the Bulk Data Section width value is to be set, i.e., if <widthinfo2> is specified. For example,

```
%defrepwidth 0.0,wide
```

specifies that default symbolic substitution is to be 0.0 before the BEGIN statement and is to be -16.16 after the BEGIN statement.

Clearing ("Unsetting") Default Width Information Using undefrepwidth

Default symbolic variable substitution width information may be cleared in a Nastran data file using the following statement:

```
%undefrepwidth
```

where the '%' character *must* be in column 1 and nothing else may appear in the record except for optional comments following the undefrepwidth string, where the start of the comment is indicated by a ' $'. The

undefrepwidth string is not case-sensitive and must be followed by at least one blank. This statement does not have any arguments and clears both default width specifications.

General Information For Special Statements

The %setrepsym, %unsetrepsym, %defrepsym, %undefrepsym, %setrepwidth, %unsetrepwidth, %defrepwidth and %undefrepwidth statements are deleted, logically, from the data file and will never be processed by the rest of Nastran unless an error is encountered while they are being processed. This is discussed in the Error Handling, 168.

Requesting Symbolic Substitution

Symbolic variable substitution will occur when a string having the form

```
%<varname>,<widthinfo>:<varvalue>%
```

is found anywhere within a Nastran data file, except that this string may *not* span records, i.e., the substitution request must be on a single record (line). The leading and trailing '%' characters are required as is the <varname> field. The <widthinfo> field is optional. If it is omitted, the comma (,) separating it from the <varname> field may be omitted and the rules for determining what width specification will be used are discussed below. The <varvalue> field is optional and provides a way of specifying a default value, i.e., the "local default value", as described below. If it is omitted, the colon (:) separating it from the <varname> (or <widthinfo>) field may be omitted. The rules for determining what symbolic value will be used as the substitution value are discussed below. For example, if the symbolic variable abc is to be replaced by its current value with no special processing (or if default width processing is to be used), the substitution request would be:

```
%abc%
```

If the symbolic variable is to be replaced by its current value, with the minimum field width to be 12 characters and with the value always to be left-justified, the substitution request would be:

```
%abc,-12%
```

Quoting Rules For Symbolic Variable Values

- If a symbolic variable value is case-sensitive, if it contains leading, trailing or embedded blanks or if it contains percent characters, tab characters or other special characters, it must be quoted. (Note that "escape" sequences such as '\t' or '\n' are not given any special treatment; that is, they are left as is.)
  - If the value is part of a repsym keyword command-line specification, the quoting rules of the command shell being used apply.
  - If the value is part of a repsym keyword specified in an RC file, it must be enclosed in single quotes (').
  - If the value is part of a %setrepsym or %defrepsym record or if it specified as the "local default value" in a symbolic substitution request, quoting a symbolic variable value means enclosing the value in one of the following pairs of characters:

Starting Quote Character	Ending Quote Character
"	"
'	'
/	/
\	\
[	]
{	}
(	)

If the first non-blank character encountered in a variable value specification is one of the starting quote characters, the variable value *must* be ended by the associated ending quote character. The actual variable value will be the (possibly null) string between (but not including) the starting and ending quote characters. If the variable value starts with one of the starting quote characters, it must be quoted using an alternate quote character.

General Rules For Symbolic Variable Substitution

- Nested symbolic substitution is not supported. Even if the value associated with a symbolic variable name is, itself, in the format of a symbolic variable substitution request, that request will be ignored. That is, after symbolic variable substitution has occurred, the substituted string is *not* re-scanned.
- Determining what symbolic variable value will be used when a variable substitution request is encountered depends on where the variable value associated with the specified variable name was set. The *first* value encountered in the following hierarchy is the value that will be used:
  - A value specified in the Nastran data file using the `%setrepsym` statement, if there is one active, i.e., if it has not been deactivated by a `%unsetrepsym` statement.
  - A value specified on the Nastran command line or in RC files using the `repsym` keyword.
  - As part of the variable symbol substitution request, using the local default value, if there is one.
  - A value specified in the Nastran data file using the `%defrepsym` statement, if there is one active, i.e., if it has not been deactivated by a `%undefrepsym` statement.

  This precedence follows normal MSC Nastran ordering, i.e., "the data file wins," while still providing great flexibility. Also, the ordering of the last two items in this hierarchy allows a user to set all defaults except for special cases and follows the idea that the specification "closest" to the use is the one used. If no replacement value is found, the substitution request will be ignored and the record will be unchanged.
- Determining what symbolic width specification will be used when a variable substitution request is encountered depends on where the width information has been specified and on the part of the Nastran data file that is being processed, i.e., is the variable substitution request before or after the first `BEGIN` statement. The *first* width specification value encountered in the following hierarchy is the specification that will be used:
  - A value specified in the symbolic substitution request itself, i.e., if a `<widthinfo>` entry was specified as part of the symbolic substitution request.

- A value specified on a `%setrepwidth` statement corresponding to the current section in the Nastran data file, if there is one active, i.e., if it has not been deactivated by an `%unsetrepwidth` statement.
- A value specified on the Nastran command line or in RC files using the `repwidth` keyword corresponding to the current section in the Nastran data file.
- A value specified in the Nastran data file using the `%defrepwidth` statement corresponding to the current section in the Nastran data file, if there is one active, i.e., if it has not been deactivated by a `%undefrepwidth` statement.
- The program default value of exact (`0.0`).

This precedence also follows normal Nastran ordering, i.e., "the record wins followed by the data file wins," while still providing great flexibility.

- When running in licensing "Interlock" mode, i.e., in CRC validation mode, the following restrictions will be in effect. If a restriction is violated, Nastran processing will be terminated.
  - The `%setrepsym`, `%unsetrepsym`, `%defrepsym` and `%undefrepsym` statements are not allowed. Also, specifying a default value within the symbolic substitution request is not allowed. That is, symbolic variable values may only be set using the `repsym` keyword on the command line or in an RC file. Note that the `%setrepwidth`, `%unsetrepwidth`, `%defrepwidth` and `%undefrepwidth` statements *are* allowed.
  - A maximum of two symbolic substitution specifications are allowed per record and a maximum of ten symbolic substitution requests are allowed in the entire input data file.
  - Interlock CRC calculations will be made on the input record *before* symbolic substitution occurs. Note that any alterations to the record made as part of the CRC calculation processing will not affect symbolic substitution processing.

Requesting Symbolic Substitution Replacement Information Using REPINFO

- A report of what symbolic substitutions were made is generated at the end of Nastran processing, with the level of detail in the report controlled by an "information level" flag set using the `repinfo=n` keyword, where *n* is an integer number that specifies the level of detail desired. The meanings the various values for n are as follows:

0	suppress the report altogether
1	report the various values assigned using the `repsym` keyword
2	same as 1 except add the various values assigned using the `setrepsym` statement
3	same as 2 except add the various values assigned using the `defprepsym` statement
4	same as 3 except add the various values assigned as local default values
5	same as 1 except add all locations where the specified `repsym` value was used
6	same as 2 and 5 except add all locations where the specified `setrepsym` value was used

7 same as 3 and 6 except add all locations where the specified `defrepsym` value was used

8 same as 4 and 7 except add all locations where local default values were used.

The report is written to the .f06 file. If there is not enough dynamic memory available to save the report information, the `repinfo` level may be reduced. When running in MSC Nastran, the default is `repinfo=1`. Otherwise, `repinfo=0` will be forced.

- Just as with other Nastran command line or RC file keywords, the `REPINFO` keyword is not case-sensitive.

Error Handling

If an error is encountered processing a `setrepsym`, `unsetrepsym`, `defrepsym`, `undefrepsym`, `setrepwidth`, `unsetrewidth`, `defrepwidth` or `undefrepwidth` statement, a comment string will be added to the record giving the error information and the record will be passed to Nastran (or the application reading the data file) as if the record was a normal Nastran data record. If an error is encountered in a record containing a symbolic substitution request, the symbolic substitution request will not be processed and, if `repinfo=1` or greater is in effect, a message giving information about the error will be written to the .log file. It is expected that the statements in error will not be valid Nastran statements and so will be flagged as an error.

Examples

1. The value on an "OPTION" statement is to be settable using the command line, taking a default value of "OPT1val" (case-sensitive) if no command line value is set. The OPTION statement could be

   ```
   OPTION=%Option:'OPT1val'%
   ```

   and the command line parameter that would be used to set OPTION to a different value, OP2VAL (not case-sensitive), would be

   ```
   RepSym=Option=op2val
   ```

2. An INCLUDE file contains records that are to be used four times in the Bulk Data Section of a Nastran data file, with the only difference being the value in Field 3 of one record. The first time the file is used, this field must contain the value 1.234, the second time this field must contain the value 4.567 and the last two times this field must contain the value -12.578. In all cases, the replacement field must be eight characters wide and the data must be left-justified in the field. Assuming that the symbolic variable is DATFL3 and that the include file name is incl.data, this could be done as follows:

   In the include file, specify the following statements before the record to be modified:

   ```
   %DefRepSym datfl3=-12.578
   ```

   then the record to be modified could be specified as follows:

   ```
   FL1      FL2      %datfl3%FL4      FL5      FL6
   ```

   and, for completeness, specify the following record after the record to be modified:

   ```
   %Undefrepsym datfl3
   ```

   Then the data file would contain:

```
. . .
%setrepsym DATFL3=1.234
%DefRepWidth  ,bulk
include 'incl.data'

. . .
%setrepsym DATFL3=4.567
include 'incl.data'
%Unsetrepsym datfl3
. . .
include 'incl.data'
. . .
include 'incl.data'
```

B Keywords and Environment Variables

- Keywords
- SYS Parameter Keywords
- Environment Variables
- Other Keywords
- System Cell Keyword Mapping

Keywords

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
	Note: This keyword can only be set in the command initialization file, see the sections titled Enabling Account ID and Accounting Data and Specifying Parameters in Appendix A.		
	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:	*nast_ver* `example after=10:00`	
	The job is held until 10:00 AM.		

(WINDOWS)	Submits the job to the Windows Scheduler. The JOB name will be "Nast_JOB". Comments: 1. The status of the job may be found with "schtasks | findstr Nast" 2. The time should be specified in 24 hours increments. Times less than 10:00 should have a preceding 0. e.g. aft=06:00. 3. The job will be left in the scheduler. You may clean with: schtasks /delete /tn Nast_JOB 4. after= is not supported with DMP jobs at this time on windows.

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:	*nast_ver* `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.		
	Note: This keyword should always be set to "NASTRAN", and may only be specified on the command line or in the command initialization file. See Specifying Parameters in Appendix A.		
attdel	attdel=*number*	Default:	0 (enables automatic assigning)
	Controls automatic assignment of the delivery database. See the Executing MSC Nastran in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
autoasgn	autoasgn=*number*	Default:	7 (all)
	Controls automatic assigning of DBsets. See the Executing MSC Nastran in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
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:	The Default is set during the installation.
	Selects the licensing method for MSC Nastran. The spec can take on several forms. They include: *authorize=FLEXlm-license-spec* FLEXlm licensing has been selected. *authorize=pathname* This specifies either 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:	*nast_ver* `example auth=myauthfile`	
	The job runs using the node-locked authorization code in "myauthfile".		
authque	authque=*number*	Default:	20
	All systems: Specifies the time in minutes to wait for a seat to become available. If the seat becomes available before this specified time period expires, the job will be allowed to continue. If not, the job will be terminated.		
	Note: When a job is waiting for a seat to become available, it consumes computer resources such as memory, swap file space, disk space, etc. Too many jobs waiting for licenses could have a severe impact on the system.		
	Example:	*nast_ver* `example auth=myauthfile`	
	The job runs using the node-locked authorization code in "myauthfile". If a seat is not available within 20 minutes of the start of the job, the job terminates.		
	Example:	*nast_ver* `example auth=myauthfile authque=10`	
	The job is run using the node-locked authorization code in "myauthfile". If a seat is not available within 10 minutes of the start of the job, the job will be terminated.		
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.		
	Note: If the job is already running in an NQS or NQE batch job, the default is "no".		
	Example:	*nast_ver* `example batch=no`	
	The job is run in the foreground.		

bfgs	bfgs=*number*	Default:	0
	Selects strategies of BFGS updates for the arc-length methods in non-linear analysis. See the Executing MSC Nastran in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
bpool	bpool=*value*	Default	*See text below.*
	Specifies the number of GINO and/or executive blocks, or memory size in MB, GB, etc., that are placed in buffer pool.		
	The size is specified as the number of blocks (BUFFSIZE words long), a percentage of MEM, or the number of words or bytes followed by one of the modifiers: "T", "TW", "TB", "G", "GW", "GB", "M", "MW", "MB", "K", "KW", "KB", "W", "B". See Specifying Memory Sizes for a description of these modifiers. If mem=max (which is the default) or/and solve=auto is not used, the default of bpool is 150 GINO blocks. If solve=auto is used, bpool will be set automatically. If mem=max is used, the default of bpool will be set to: ■ 25% of memory for non SOL 101 or SOL 400. ■ the remaining memory after memory estimate needed for the solver for SOL 101 or SOL 400.		
	Example:		*nast_ver* `example bpool=100mb`
buffpool	buffpool=*number*	Default:	*See* bpool
	Specifies the number of GINO and/or executive blocks, or memory in MB, GB, etc., that are placed in the buffer pool. This keyword is a synonym for the "bpool" keyword. See the description of the "bpool" keyword for more information.		

buffsize	buffsize=*value*	Default:	32769
	Specifies the physical record size, in words (32769 * 8 bytes), of all MSC 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:	*nast_ver* `example buffsize=16385`	
	The BUFFSIZE is set to 16385 words.		
casi	If set to "no" it will disable the casi solver as a possible option when "solve=auto" is specified.		
childout	childout=yes,no	Default:	No
	Specifies the output files from the child nodes are to be copied back to the local node.		
config	config=*number*	Default:	Computer dependent
	Specifies the configuration (CONFIG) number used by MSC 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.		
constitle (Windows)	constitle=yes, no	Default:	Yes
	Specifies whether or not the console title bar is to be modified to have the job identification. This keyword is only applicable to Windows systems.		

cpumax	cpumax cpumax=*n*	Default:	1
	Selects the best SMP and DMP combinations for SOL 101, 103, 107, 108,111, 200 and 400 based on the solver (matrix solver of MSCLDL, PARDISO, CASI, or modal solver of Lanczos, ACMS) selected and limit the number of cores ((#DMP)(**#SMP) ≤ n). This option is usually used together with solve=auto but can be used without solve=auto. If solve=auto is used without cpumax, then cpumax is the number of processors on the system (of one node). If solve=auto is used with cpumax = n, it sets the limit for the number of cores** ((**#DMP**)(#SMP) ≤ n). The cpumax may be set on the command line or in a user's RC file. If neither cpumax or solve=auto is used, the cores of #SMP or/and #DMP are dependent on the smp or/and dmp selection. If none of cpumax, solve=auto, smp, dmp or their combination is used, the job will be run serially without using parallel.		
cputime (LINUX)	cputime=*cputime*	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 MSC Nastran executable, plus any commands specified by the "pre" and "post" keywords. See your system's queuing documentation for the format of *cputime*. 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:	*nast_ver* example queue=small cputime=60	
	This example defines the maximum CPU time for the complete job as 60 seconds.		
	Example:	*nast_ver* example queue=small cpu=1:15:0 *nast_ver* example queue=small cpu=75:0 *nast_ver* example queue=small cpu=4500	
	These examples all define the maximum CPU time for the complete job as one hour and fifteen minutes.		
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:	*prod_ver* `estimate example dballco=2`	
	This will double the DBALL disk estimate and then apply the "dballmin" lower bound.		
	Example:	*prod_ver* `estimate example dballco=0.5`	
	This will halve the DBALL disk estimate. An estimate less 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:	*prod_ver* `estimate example dballmin=2mb`	
	This will set the minimum DBALL disk estimate to 2 MB.		
dbs	dbs=*pathname*	Default:	.
	Creates database files (see Table 4-7) 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:	*nast_ver* `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:	*nast_ver* `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:	*nast_ver* `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:	*nast_ver* `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:	*nast_ver* `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:	*nast_ver* `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: The use of distinct per-task database directories can have a significant impact on elapsed time performance of DMP jobs on SMP and NUMA systems.		
dbverchk	dbverchk=0, 1	Default:	0 (check is performed)
	Specifies whether or not database version checking is to be skipped. See the Executing MSC Nastran in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
delete	delete=yes, no, all, jid, *list*	Default:	No
Note:	This keyword is only intended to be used when MSC Nastran is running as a server or is embedded within an other application. The deletion occurs before the post commands are run.		
	Unconditionally delete files after an MSC Nastran job completes. Specifying "delete=yes" will delete the F04, F06 and LOG files 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. Note that, on LINUX systems, this list of file types is *case-sensitive*. That is, "delete=master" will delete files with an extension of "master" but not files with an extension of "MASTER" and "delete=MASTER" will delete files with an extension of "MASTER" but not files with an extension of "master".		
	Example:	*nast_ver* `example delete=op2,plt`	
	After the MSC 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.		
	Example:	*nast_ver* `example delete=plt,MASTER,DBALL`	
	After the MSC Nastran job has completed, the "example.plt", "example.MASTER" and "example.DBALL" files will be unconditionally deleted. Normally, the "example.plt" file will kept if it is not empty and the "example.MASTER" and "example.DBALL" files are kept unless "scratch=yes" was specified.		

delivery	delivery=*pathname*	Default:	MSCDEF
	Specifies an alternate delivery database option. See Creating and Attaching Alternate Delivery Databases for further information on alternate delivery databases. The special pathname “MSCDEF” indicates the standard MSC Nastran delivery database.		
	Example:	*nast_ver* `example del=mysss`	
	The job runs using a solution sequence from the delivery database “mysss.MASTERA”.		
diag	diag=*flag,flag,...*	Default:	None
	Sets MSC Nastran diagnostics. This keyword may also be set with the DIAG Executive Control Statement. See DIAG (p. 124) in the *MSC 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:	*nast_ver* `example diag=5`	
	The MSC Nastran job is run with DIAG 5 set.		
diaga	diaga=*number*	Default:	None
	Set MSC Nastran diagnostic flags 1-32. The value specified over-rides any previous "diag=n" values where n is in the range 1 to 32. These diagnostics are set in addition to any diagnostics set via the Executive Control "DIAG" statement in the input data file. See the Executing MSC Nastran in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
diagb	diagb=*number*	Default:	None
	Set MSC Nastran diagnostic flags 33-64. The value specified over-rides any previous "diag=n" values where n is in the range 33 to 64. These diagnostics are set in addition to any diagnostics set via the Executive Control "DIAG" statement in the input data file. See the Executing MSC Nastran in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
disksave	disksave=*number*	Default:	0 (no save)
	Specifies Lanczos High Performance Option controlling whether or not the matrix/vector multiply is saved in a scratch file. See the Executing MSC Nastran in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
distort	distort=*number*	Default:	0 (terminate run)
	Specifies element distortion fatal termination override. Applies to all p-elements and the TETRA h-elements. See the Executing MSC Nastran in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		

dmparallel (See Table 5-2)	dmparallel=*number*	Default:	0
	Specifies the number of tasks for a Distributed Memory Parallel (DMP) analysis. The value must be null or zero to cancel DMP processing, or a number greater than zero to enable DMP processing. See Running Distributed Memory Parallel (DMP) Jobs for additional information.		
	Example:	*nast_ver* `example dmp=4`	
	The job is run with four DMP tasks.		
dmpmem	dmpmem=*number*	Default:	80
	Specifies the percentage of the total memory specified by memorymax for a DMP job to be allocated to the Parent DMP process. The dmpmem keyword is applicable only for DMP jobs that use ACMS (VERSION=NEW). The percentage of memory not given to the Parent DMP process by dmpmem is split evenly among the remaining DMP processes. The value of dmpmem must be between 51 and 95. The default value of dmpmem is 80. By default, 80% of the total memory specified by memorymax and mem=max is reserved for the Parent DMP process. The remaining 20% is reserved for the Child DMP processes.		
dranas_nast_mem	dranas_nast_mem=*value*	Default=*2048mb*	
	Defines the memory for MSC Nastran Server. In case of large data base if Patran cannot attach to the MSC Nastran data base then the cause could be that the server has run out of open core memory. In such a case, increasing the value of the variable could solve the problem.		
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:	*prod_ver* `estimate example dskco=2`	
	This doubles the total disk estimate and then applies the "dskmin" lower bound.		
	Example:	*prod_ver* `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:	*prod_ver* `estimate example dskmin=2mb`	

	This will set the minimum total disk estimate to 2 MB.		
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:	*nast_ver* `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*/*prod_ver*/*arch* on LINUX or *install_dir**prod_ver**arch* on Windows.		
expjid	expjid=no, yes, auto, *pathname*	Default:	Auto
	Specifies whether or not the input file is to be "expanded" or not, that is, whether or not the input file is to be read and all "include" files processed. If "expjid=no" is specified, the input file will be used directly.		
	If "expjid=yes" is specified, the input file will be expanded and stored in the location specified by "out", with an extension of "exp" added.		
	If "expjid=*pathname*" is specified, the input file will be expanded and stored in the location specified by *pathname*. If *pathname* specifies a directory, the expanded file will be stored using the base name of the input file, with an extension of "exp" added. If *pathname* specifies a file name without an extension, and extension of "exp" will be added.		
	If "expjid=auto" is specified (or taken by default): ■ If "node" is specified, the input file will be expanded only if it is not visible from the remote node. ■ If "node" is not specified, the input file will not be expanded, i.e., processing will be as if "expjid=no" was specified.		
	If the input file is expanded: ■ If "node" is specified, the expanded file will be copied (if necessary) to the remote node for processing and will be deleted from both the remote and local nodes at the completion of processing. ■ If "node" is not specified, processing will terminate without actually invoking the MSC Nastran analysis program and without storing any other files.		

extdefault	extdefault=logname1(ext1),logname2(ext2),... Default: None		
	Changes the default extension(s) associated with the specified logical name(s). The logical name (lognamei) specification is case-insensitive. For LINUX/Linux systems, the extension (exti) is casesensitive; for Windows systems, it is not. This keyword may be specified as often as necessary and the values will be comma-separated and appended. If the same logical name is specified more than once, the extension in the last specification will be the one used. `Example: extdefault=plot(myplot),punch(mypunch)` changes the default extension assigned to logical name PLOT from "plt" to "myplot" and the default extension assigned to logical name PUNCH from "pch" to "mypunch". Restrictions: 1. Only FORTRAN files identified as "Assignable" may have their default extension changed . See Table 2-1 in the MSC Nastran Quick Reference Guide, Volume 1. 2. The extension may not be one of the reserved extensions. These reserved extensions are: f06, f04, log, MASTER, DBALL, OBJSCR, SCRATCH and SCR300. 3. Extensions must be from one- to eight-characters long.		
f04	f04=*number*	Default:	4
	Specifies FORTRAN unit number for Execution Summary Table. See the Executing MSC Nastran in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
f06	f06=*number*	Default:	6
	Specifies FORTRAN unit number for standard output file. See the Executing MSC Nastran in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
fbsmem	fbsmem=*number*	Default = 0	
	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 MSC Nastran Quick Reference Guide for information 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. See the *MSC Nastran Version 69 Release Guide* for additional information on external bar or beam elements. Also, see Using BEAMSERV for information on running an MSC Nastran job using a beam server.		

	Example:	*nast_ver* `example gmconn=mybeamserver`	
	The job is run with the external evaluators specified in “mybeamserver”.		
gpuid	gpuid=id,id or gpuid=id:id		Default: none
	id: the ID of a licensed GPGPU device to be used in the analysis. Multiple IDs may be assigned to Nastran distributed memory processor (DMP) runs. Separate a list of IDs with a comma or a colon. Each DMP process will be assigned a GPU ID in round robin fashion.		
gpu_min_front	gpu_min_front=value		Default: 16
	The criteria for GPGPU execution during matrix factorization are the frontal matrix front size and the rank of the frontal matrix. Minimum dimensions are set via gpu_min_front and its companion parameter, gpu_min_rank. The value specified must be an integer greater or equal to 1. If the front size is smaller than value, the rank update of the front is processed on the CPU. Otherwise, the GPGPU device would be used for the rank update of the front. This keyword may also be set via SYSTEM cell 656.		
gpu_min_rank	gpu_min_rank=value		Default: 1024
	The criteria for GPGPU execution during matrix factorization are the frontal matrix front size and the rank of the frontal matrix. Minimum dimensions are set via gpu_min_front and its companion parameter, gpu_min_rank. The value specified must be an integer greater or equal to 1. If the rank of the frontal matrix is smaller than value, the rank update of the front is processed on the CPU. Otherwise, the GPGPU device would be used for the rank update of the front. This keyword may also be set via SYSTEM cell 655.		
hicore	hicore=*memory_size*	Default:	*Dependent on "memory" and other keywords*
	Specifies maximum working memory. See the Executing MSC Nastran in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
hostovercommit	hostovercommit=yes,no	Default:	No
	Allows this job to assign more tasks to a host than processors. This does **not** prevent other MSC 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:	*nast_ver* `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 host.		
	Example:	*nast_ver* `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	hosts=*host*:*host*:... hosts=*host*;*host*;... hosts=*filename*	Default:	See text
	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.		
	The default is the current system. See Running 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:	*nast_ver* `example dmp=2`	
	The job will be run on the current host.		
	Example:	*nast_ver* `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:	*nast_ver* `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.		
	Some linux systems may require the setting when running on multiple hosts: `export I_MPI_HYDRA_BOOTSTRAP=ssh`		
ifpbuff	ifpbuff=value	Default:	4096

	Specifies the physical record size, in words, of MSC Nastran IFPStar database. The physical I/O size is IFPBUFF-1 words. The maximum value of IFPBUFF is 65537 words.		
	Example:	prodver nastran example ifpbuff=8193 The IFPBUFF is set to 8193 words.	
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 MSC 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. Where TPLDIR is the directory that contains the files in the tpl or tpl6 directories in the seperate documentation/ example problem installer. On Windows, it may be necessary to define "CMD.EXE" as the processor for certain ".EXE" files. This can be done with "ishellext=exe=cmd" Up to twenty associations can be defined.	
	This keyword may also be set with the *MSC_ISHELLEXT* environment variable. The environment variable overrides the RC files; the command line overrides the environment variable.	
	Example:	*nast_ver* `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 *MSC_ISHELLPATH* environment variable. The environment variable overrides the RC files; the command line overrides the environment variable.		
	Example:	*nast_ver* `TPLDIR:qaishell`	
	Assuming no RC file set "ishellpath" and the environment variable MSC_ISHELLPATH was not defined, the "ishellpath" value will be set to the directory referenced by "TPLDIR:". MSC Nastran will attempt to locate the ISHELL program in the current working directory, the TPL directory, or in the PATH.		
	Example:	*prod_ver* `example ishellpath=bin`	
	This example assumes either the current working directory or the bin subdirectory contains the ISHELL program		
iter	iter=yes, no	Default:	No (do not execute iterative solver)
	Controls execution of iterative solver. See the Executing MSC Nastran in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
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:	*nast_ver* `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. One or more of these directories may include a "wildcard" specification, as described below, requesting sub-directory searching. On LINUX, the directory level separator character is slash ("/"). On Windows , the directory level separator character can be slash ("/") or back-slash ("\"). This keyword may also be set by the MSC_JIDPATH environment variable. The environment variable overrides the RC files, and the command line overrides the environment variable. This keyword may not be specified in a conditional section of an RC file. It is very important to note that the directory list specified by this keyword is *not* cumulative. That is, the directory list specified by this keyword completely replaces the directory list specified by a previous instance of this keyword. However, see the description below about environment variable replacement for an example of how a cumulative definition could be specified.		
	LINUX example:	*nast_ver* `example jidpath=$HOME`	
	Windows Example:	*nast_ver* `example jidpath=%HOMEDRIVE%%HOMEPATH%`	
	These will 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. Sub-directory searching is requested by specifying the "wildcard" character ("*") as the last directory component of a search path directory. If sub-directory searching is requested, the directory specification up to (but not including) the wildcard specification and *all* of its sub-directories will be searched for the input data file. Note: If there are multiple files in the sub-directories with the same filename, it is unpredictable which file will be located.		
Notes:	On LINUX, the "tilde" capability is supported. That is, if a directory in the list starts with a tilde ("~"), it will be replaced with the home directory of the current user if the tilde is the only character in the directory or if the tilde is followed by a "/". If the tilde is followed by a character string, i.e., has the form "~*name*", the entire "~*name*" specification be replaced by the home directory of user "*name*". If the home directory information cannot be determined, the directory will be skipped. On LINUX, jidpath specifications on the command line that include any sub-directory searching requests should be enclosed in quotes to prevent the Shell from expanding the wildcard character. Quoting generally is not required for Windows.		

	For example:		
	LINUX example:	*nast_ver* `example jidpath=/models/a:/models/b`	
	Windows Example:	*nast_ver* `example jidpath=\models\a;\models\b`	
	If sub-directory searching is to be enabled for the second directory, the specification could be:		
	LINUX example:	*nast_ver* `example jidpath="/models/a:/models/b/*"`	
	Windows Example:	*nast_ver* `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. This method, for example, could be used to implement a "cumulative" search path specification
	
	For example, if the current keyword value (e.g., `"/new/path"`) is to be added to the search path prior to the previous value, on LINUX, the keyword could be specified as:
	
	`jidpath=/new/path:$MSC_JIDPATH`
	
	and if it is to be added at the end of the previous value, on Windows, the keyword could be specified as:
	
	`jidpath=%MSC_JIDPATH%;/new/path`		

jidtype	jidtype=*file-type*	Default:	dat
	Specify an alternate default file-type of the input data file and any INCLUDE files.		
	Example:	*nast_ver* `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:	*nast_ver* `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".		
ldqrkd	ldqrkd=*number*	Default:	0 (Version 68+ method)

	Selects the differential method for CQUAD4 and CTRIA3 elements. See the Executing MSC Nastran in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
locbulk	locbulk=*number*	Default:	0 (RESTART FMS statement)
	Specifies that special Bulk Data processing is in effect. See the Executing MSC Nastran in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
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 RC File Keywords in Appendix A 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 “MSC_LICENSE_FILE” or “LM_LICENSE_FILE”, or on the command line will be silently ignored.		
lsymbol	lsymbol=*name=string*	Default:	None
	This keyword has the same general function and syntax as the "symbol" keyword except that it defines a "local" symbolic (or logical) name. Symbols defined using this keyword will not be passed to remote hosts, i.e., to hosts specified by the "node" keyword. When the "node" keyword is not specified, this keyword is synonymous with the "symbol" keyword.		
maxlines	maxlines=*number*	Default:	999999999
	Specifies the maximum number of output lines. See the Executing MSC Nastran in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
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:	*prod_ver* `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:	max
	Specifies the amount of memory to allocate. If "memory=estimate" is specified, ESTIMATE will be used to determine size. If "memory = max" is specified. memory will be set to *memorymaximum* (divided by number of DMP processors per node). bpool will be set to: ■ 25% of MEM for non SOL 101 or SOL 400. ■ the remaining memory after memory estimate needed for the solver for SOL 101 or SOL 400. Otherwise, the *size* is specified as a memory size, see Specifying Memory Sizes. If a value was not assigned to the "memory" keyword, or if "memory=estimate" was specified and ESTIMATE failed to provide an estimate, the nastran command will use the value specified by the "memorydefault" keyword. If the "memorydefault" value is null, the nastran command will issue a fatal error and the job will end. Example:		
	nast_ver `example memory=200MB`		
	The job is run using a memory size of 25 MW, or 25600 KW, or 26214400 words. Example:		
	nast_ver `example memory=0.5xPhysical`		
	If run on Windows, the job is run using a 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:	8mw
	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:	0.5*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 MSC 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:	*nast_ver* `example memory=900mb`	
	The job is run using a memory size of 900MB.		
	Example:	*nast_ver* `example memory=1200mb`	
	The job is run using a memory size of 1GB, i.e., the "memorymaximum" value set in the RC file. If multiple Nastran jobs are running simultaneously on the same node of a compute system, it is recommended to set memorymax to 0.8*physical_memory divided by the number of possible Nastran jobs running on the same node. Otherwise, one might encounter an error due to lack of memory on small models. The error can be avoided by using above setting in the system RC file or mem=size specified properly for the job in the command line.		
mergeresults	mergeresults=yes,no	Default:	Yes
	Specifies the results from each DMP task are to be merged into the standard files from the parent host. Setting "mergeresults=yes" will cause the output from all tasks to appear in the output files for the parent 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 parent 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 Patran in solution 108. In solution 108, if "mergeresults=no" is specified and "childout=yes" is not specified, then the results of the child processors will be lost. In solution 108, it is possible to get a through-put advantage by saving communication between the parent and children when "mergeresults=no" and "childout=yes" is specified.		
metime	metime=*number*	Default:	-1

	Minimum time for execution summary table message. See the Executing MSC Nastran in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
mindef	mindef=*number*	Default:	1 (do not check)
	Indefinite Mass Matrix Check flag. See the Executing MSC Nastran in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
minfront	minfront=*number*	Default:	Machine dependent
	Set the rank minimum front size in the sparse modules. See the Executing MSC Nastran in the *MSC Nastran Quick Reference Guide* for more information on this keyword. This value may also be set with the "rank" keyword.		
mode	Deprecated. MODE =i8 is set running Installation to I8. MODE=I4 is no longer supported.		
mperturb	mpreturb=*number*	Default:	1 (do not perturb)
	Set the perturbation factor for indefinite mass matrix. See the Executing MSC Nastran in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
mpyad	mpyad=*number*	Default:	See the MSC Nastran Quick Reference Guide.
	Selects/deselects multiplication method selection. This keyword may also be set with the "sys66" command line keyword. See the MSC Nastran Quick Reference Guide for information on the default value and legal values for this keyword.		
msgbell	msgbell=yes, no, bell	Default:	Yes
	Specifies whether or not the job completion string will include an audible message ("bell" sound) or not. "Yes" or "bell" says that three "bell" sounds will be appended to the job completion string. "No" suppresses the bell sounds.		

msgcat	msgcat=*pathname*	Default:	LINUX: *install_dir*/*prod_ver*/*arch*/ analysis.msg Windows: *install_dir**prod_ver**arch*\ analysis.msg
	The "msgcat" keyword specifies an alternate message catalog containing the message text used for many MSC 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:	*nast_ver* `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.		
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 “MSC Nastran job *name* completed”.		
	Example:	*nast_ver* `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 "MSC/NASTRAN job example completed" | mail -s "MSC/NASTRAN job example completed" user`		
	Windows example:	*nast_ver* `example "ncmd=echo done"`	
	The word “done” will be printed in the command window when the job completes.		
newhess	newhess=*number*	Default:	See the MSC 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 *MSC Nastran Quick Reference Guide*, and the *MSC 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*/*prod_ver*/nast/news.txt on LINUX and *install_dir**prod_ver*\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:	*nast_ver* `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: *nast_ver* `news`		
nlines	nlines=*number*	Default:	50
	Specifies number of lines printed per page of output. See the Executing MSC Nastran in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
node	node=*nodename*	Default:	None
	Executes the job on the specified node. See Running a Job on a Remote System, 103 for additional information. This keyword may only be specified on the command line.		
	Use the "username" keyword to specify an alternate user name on the remote node.		
	Example:	*nast_ver* `nastran example node=othernode`	
	The job is run on the computer named "othernode".		
	If the remote node is a LINUX node, ssh/scp processing must be enabled.		
	If the remote node is a Windows node running Window 7 or 10, the MSCRmtMgr program must be running on that node, either as a started service or as a program running in a command prompt window.		
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:	*nast_ver* `example notify=yes`	
nsegadd	nsegadd=*number*	Default:	2
	Number of segments in the element error table in adaptive analysis. See the Executing MSC Nastran (p. 1) in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
numseg	numseg=*number*	Default:	See text.
	Sets the number of segments for the Lanczos High Performance Option. See EIGRL (p. 1755) in the *MSC Nastran Quick Reference Guide* for information on the default value and legal values for this keyword.		
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:	*nast_ver* `example old=yes`	
	For example, assume your current working directory contains the following files:		

	`v2401.dat v2401.f04 v2401.f06 v2401.log`
	`v2401.f04.1 v2401.f06.1 v2401.log.1`
	`v2401.f04.2 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:	*nast_ver* `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 MSC_OLDTYPES environment variable. The environment variable overrides the RC files, and the command line overrides the environment variable.		
Intel oneAPI	Intel oneAPI=true,false	Default:	false
	UDS jobs will use Intel oneAPI compilers when applicable. Please see Using the Advanced Functions of MSC Nastran for details. This option may not be set in RC files.		
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:	*nast_ver* `example`
	or:	*nast_ver* `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:	*nast_ver* `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:	*nast_ver* `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:	*nast_ver* `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
	Specify the Number of processors used for Shared Memory Parallel		
	Example:	*nast_ver* `example parallel=2`	
	The job is run in SMP mode on a maximum of two cores.		

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. Commands with spaces in the path, arguments or linux pipes should be put in a single command file and "post=" should point to that command. For example, consider the windows batch script in a directory with spaces and command line arguments:
	`D:\scratch>type "a b\c.bat"`
	`echo off`
	`echo in c.bat`
	`echo %1`
	There are spaces in the path and the command has arguments, resulting in additional spaces. To execute, create a command file with appropriate quotes:
	`D:\scratch>type cmd.bat`
	`"a b\c.bat" x`
	This command can then be run with "post=cmd.bat":
	Similar structures may be used on Linux with pipes, wildcards, etc.		

Note:	In order 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 MSC 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:	*nast_ver* `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 MSC 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. (Altix systems also include EAG FFIO cache).		
	The *size* is specified as a memory size, see Specifying Memory Sizes, 63.		
	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:	*nast_ver* `example queue=small mem=100m ppmdelta=10m`	
	The job is submitted to the small queue with a memory size of 100 MW, and a per-process memory limit of 110 MW.		
pre	pre=*command_string*	Default:	None
	Runs the specified command before the job begins. Commands with spaces in the path, arguments or linux pipes should be put in a single command file and "pre=" should point to that command. See post for examples		
	See Environment Variables, 217, 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 MSC Nastran process plus any other concurrent or parent processes. The minimum value is 5120 words = 5120*8 bytes. The *size* is specified as a memory size, see Specifying Memory Sizes, 63.		
	Example:	*nast_ver* `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, MSC Nastran provides more than one executable. The baseline executable has the filename "analysis" on LINUX and "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.		
punch	punch=*number*	Default:	7
	Specifies FORTRAN unit number for PUNCH file. See the Executing MSC Nastran (p. 1) in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
q4skew	q4skew=*number*	Default:	30.0
	Minimum allowable value of skew for the CQUAD4 element. See the Executing MSC Nastran (p. 1) in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
q4taper	q4taper=*number*	Default:	30.0
	Maximum allowable value of taper for the CQUAD4 element. See the Executing MSC Nastran (p. 1) in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
qclass (LINUX)	qclass=*string*	Default:	None
	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:	*nast_ver* `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.		
quadint	quadint=*number*	Default:	0 (quadratic)
	Specifies quadratic or linear interpolation for line search method in nonlinear analysis. See the Executing MSC Nastran (p. 1) in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
queue	queue=*string*	Default:	None
(LINUX)			
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:	*nast_ver* `example queue=small`	
	This example submits the job to the small queue.		
radlst	radlst=*number*	Default:	0
	Print radiation area summary. See the Executing MSC Nastran (p. 1) in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
radmtx	radmtx=*number*	Default:	0
	Type of radiation exchange coefficients. See the Executing MSC Nastran (p. 1) in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
rank	rank=*number*	Default:	See System Descriptions, 227
	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, $SYSTEM(205) \geq SYSTEM(198)$. 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 3-5 for the default values of these system cells.		
rcf	rcf=*pathname*	Default:	None

	Specifies the name of the local RC file. If this keyword is not specified, the local RC files located in the input data file's directory are used. See Command Initialization and Runtime Configuration Files, 138 in Appendix A for information about the names of these files.		
	Example:	*nast_ver* `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 files.		
rcmd (LINUX)	rcmd=*pathname*	Default:	See text.
	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:	*nast_ver* `example rcmd=/msc/bin/nast20231`	
	The pathname of the nastran command on the remote system is explicitly defined as /msc/bin/nast20231. If this file does not exist, or is otherwise not executable, the job will fail.		
rdbs	rdbs=*pathname_prefix*	Default:	.
	Remote Node alternate user database prefix. Overrides "scratch=yes" and "dbs=". If the prefix is a directory, 'jid-basename' is appended. The default on the remote node is "dbs=./'jid-basename'". This keyword is ignored unless "node=" is specified.		
rdelivery	rdelivery=*pathname*, MSCDEF	Default:	MSCDEF
	Remote Node alternate delivery database prefix or "MSCDEF". This keyword overrides all MSC-supplied solution sequences. See Creating and Attaching Alternate Delivery Databases, 131 for further information on alternate delivery databases. If a directory is not specified, the default delivery database directory is assumed. The default is "rdelivery=MSCDEF. This keyword is ignored unless "node=" is specified.		
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 MSC Nastran Quick Reference Guide for further information. The *size* is specified as a memory size, see Specifying Memory Sizes, 63. On LINUX systems, the default is "0". On Windows systems, the default is calculated using "realdelta".		

realdelta (Windows)	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, 63. 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:	*nast_ver* `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).		
repinfo	repinfo=*info_value*	Default:	1
	Specifies the Symbolic Substitution report level. This information is printed at the end of the .f06 file, after the "END OF JOB" record. This keyword may be specified in initialization or RC files and on the command line. A value of 0 suppresses the report entirely. The maximum value is 8.		
repsym	repsym=*name*=*string*	Default:	None
	Defines a Symbolic Substitution variable name and value. This keyword may be specified in initialization or RC files and on the command line. The symbol definition may include references to previously defined symbols or environment variables using the standard "$*name*" or "${*name*}" syntax on LINUX or %*name*% syntax on Windows. For convenience, the character separating the "repsym" and "*name*" specification and the "*name*" and "*string*" specification may be either an equal sign ("=") or a hash mark ("#"). The use of a hash mark allows this keyword to be specified as an argument to a Windows .bat file.		
	Symbolic Substitution names are processed in the order they are encountered while processing the initialization and RC files and the command line. If a duplicate name is encountered, the new value replaces the previously specified value.		
	Symbolic names must be 16 characters or less, the value assigned to the symbolic name must be 256 characters or less.		
repwidth	repwidth=*width_spec*	Default:	exact,exact
	Specifies the width information to be used in Symbolic Symbol substitution. This keyword may be specified in initialization or RC files and on the command line.		
rexecutable	rexecutable=*pathname*	Default:	Computer dependent
	Remote Node alternate solver executable. This keyword overrides all architecture and processor selection logic. If a directory is not specified, the default executable directory is assumed. This keyword is ignored unless "node=" is specified.		
rgmconn	rgmconn=*pathname*	Default:	None

	Remote Node Geometric evaluator connection file. See the description of the "gmconn" keyword for more detailed information. This keyword is ignored unless "node=" is specified.		
rmsgcat	rmsgcag=*pathname*	Default:	See msgcat= keyword
	Remote Node binary message catalog path name. If a directory is not specified, the default executable directory is assumed. This keyword is ignored unless "node=" is specified.		

rdebug	rdebug=yes, no, number	Default:	Yes
	This keyword controls what, if any, debug settings ("-d" options) are propagated to a remote node. "Yes" will send all current debug flags (except for "SHELL" and "RSHELL"), "no" will not pass any current debug flags. Specifying a number will set the remote debug flags to that value, where a value of "0" is equivalent to "no". This keyword is ignored unless "node" is specified.		
rostype	rostype=windows, 1, linux, 2	Default: None	
	Specifies the remote node operating system type. "Windows" and "1" are equivalent. "Linux" and "2" are equivalent. If this keyword is not specified, the nastran command will attempt to determine the remote node operating system type dynamically. This type code is used to determine the format of the remote commands used, for example, to test for file existence or to delete temporary files on the remote node. Also, if "rrmtuse" is not specified, this keyword will determine what communications programs are used, where "Windows" is equivalent to "rrmtuse=mscrmtcmd" and "Linux" is equivalent to "rrmtuse=ssh". This keyword is ignored unless "node" is specified.		
rrmtuse	rmtuse=mscrmtcmd, 1, ssh, 2	Default: None	
	Specifies which communications programs are to be used to access the remote node. "mscrmtcmd" and "1" are equivalent. "ssh" and "2" are equivalent. If "ssh" is specified, the remote node will be assumed to be a LINUX system. If this keyword is not specified and if the "rostype" keyword is specified, "mscrmtcmd" will be assumed if the "rostype" value is "windows" and "ssh" will be assumed if the "rostype" value is "unix". This keyword is ignored unless "node" is specified.		
rsdirectory	rsdirectory=*pathname*	Default:	See sdirectory= keyword
	Remote Node directory for scratch files. This is the default directory for user database files if "scratch=yes". If this keyword is not specified, the "sdirectory" value is used. Please see the description of the "sdirectory" keyword for the default value. This keyword is ignored unless "node=" is specified.		
rtimeout	rtimeout=*number*	Default:	60
	Specifies the timeout value, in seconds, to be used by "mscrmtcmd" (or the program defined by the "s.rmtcmd" keyword) in accessing a remote node. This keyword is ignored unless "node" is specified.		

s.rmtcmd	s.rmtcmd=*pathname*	Default:	mscrmtcmd
	Specifies the full pathname to the MSC Remote command used to communicate with Windows or Linux systems. This keyword may only be specified in the Initialization file or on the command line. This keyword is ignored unless "node" is specified.		
scr300	scr300=*number*	Default:	2 (create)
	Requests creation of SCR300 partition on SCRATCH DBset. See the Executing MSC Nastran (p. 1) in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
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:	*prod_ver* `estimate example scr300co=2`	
	This will double the SCR300 disk estimate and then apply the "scr300min" lower bound.		
	Example:	*prod_ver* `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.		
scr300del	scr300del=*number*	Default:	100
	Sets minimum number of blocks of SCR300 partition of SCRATCH DB set at which it is deleted. See the Executing MSC Nastran (p. 1) in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
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:	*prod_ver* `estimate example scr300min=2mb`	
	This will set the minimum SCR300 disk estimate to 2 MB.		

scratch	scratch=yes,no,mini,post	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 12: Interface With Other Programs of the *MSC Nastran Reference Guide* for further details on the “mini” database. If scratch=post is specified, a reduced size database intended for use by Patran or the toolkit will be created. Scratch=post also performs the actions of NASTRAN INDEX=19.		
	Example:	*nast_ver* `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).		
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:	*prod_ver* `estimate example scratchco=2`	
	This will double the SCRATCH disk estimate and then apply the "scratchmin" lower bound.		
	Example:	*prod_ver* `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:	*prod_ver* `estimate example scratchmin=2mb`	
	This will set the minimum SCRATCH disk estimate to 2 MB.		
scrsave	scrsave=*number*	Default:	0 (do not reuse)
	Lanczos High Performance Option that controls reuse of scratch files in segment logic. See the Executing MSC Nastran (p. 1) in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		

sdball	sdball=*size*	Default:	Computer dependent
	Specifies an alternate default size for the DBALL DBset. The computer-dependent default is listed in Computer Dependent Defaults, 230. 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 or bytes followed by one of the modifiers: "T", "TW", "TB", "G", "GW", "GB", "M", "MW", "MB", "K", "KW", "KB", "W", "B". See "Specifying Memory Sizes, 63 for a description of these modifiers.		
Note:	No attempt is made to verify if the DBALL DBset can ever grow to the size specified by this keyword.		
	Example:	*nast_ver* `example sdball=1024gb`	
	Defines the default size of the DBALL DBset as 1TB.		
	Example:	*nast_ver* `example sdball=.5tb`	
	Defines the default size of the DBALL DBset as .5TB or 512GB.		
sdirectory	sdirectory=*directory*	Default:	*See text.*
Note:	See Determining System Limits, 44 for information on estimating a job's total disk space requirements.		
	Specifies the directory to use for temporary scratch files created during the run. MSC 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:	*nast_ver* `example sdir=/scratch`	
	Scratch files are created in the /scratch directory.		
	Windows example:	*nast_ver* `example sdir=d:\scratch`	
	Scratch files are created in the d:\scratch directory		
	If a DMP run was selected with $dmparallel \geq 1$, 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. See Running Distributed Memory Parallel (DMP) Jobs, 111 for additional information.		

	LINUX example:	*nast_ver* `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.		
smaster	smaster=*size*	Default:	Computer dependent
	Specifies an alternate default size for the MASTER DBset. The computer-dependent default is listed in Computer Dependent Defaults, 230. 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 or bytes followed by one of the modifiers: "T", "TW", "TB", "G", "GW", "GB", "M", "MW", "MB", "K", "KW", "KB", "W", "B". See Specifying Memory Sizes, 63 for a description of these modifiers.		
Note:	No attempt is made to verify if the MASTER DBset can ever grow to the size specified by this keyword.		
	Example:	*nast_ver* `example smaster=1024gb`	
	Defines the default size of the MASTER DBset as 1TB.		
	Example:	*nast_ver* `example smaster=.5tb`	
	Defines the default size of the MASTER DBset as .5TB or 512GB.		
smemory	smemory=*value*	Default:	100
	Specifies the amount of space in memory to reserve for scratch memory. The size is specified as the number of blocks (BUFFSIZE words long) or the number of words or bytes followed by one of the modifiers: "T", "TW", "TB", "G", "GW", "GB", "M", "MW", "MB", "K", "KW", "KB", "W", "B". See Specifying Memory Sizes, 63 for a description of these modifiers. The value specified using this keyword may be overridden by the FMS statement INIT SCRATCH (MEM=value).		
	Example:	*nast_ver* `example smem=200`	
	This example reserves 200 GINO blocks for scratch memory.		
	Example:	*nast_ver* `example smem=4mw`	
	This example reserves 4,194,304 words for scratch memory.		
	Example:	*nast_ver* `example smem=2.5mw`	
	This example reserves 2,621,440 words for scratch memory.		
smp	smp=*value*	Default:	0
	Specifies the maximum number of processors 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 “smp=1", the parallel algorithms are used on one processor.		

Note:	If you need to vary the number of SMP processors during a job, you must set either the "smp" keyword or SYSTEM(107) on a MSC NASTRAN statement to the maximum number of SMP processors that will be requested. Some systems cannot process a DMAP request for processors in excess of this initial value.		
solve	solve=*option*	Default:	None
	solve=auto will automatically set the memory, cores, and solver based on your model and available hardware. This option may be put on the command line or User's RC files. Solve=train will train a system for memory required for the Pardiso sovler. See the HPC Guide for more information. SOLVE=AUTO may under predict memory for models with CPYRAM elements.		
sparse	sparse=*number*	Default:	See QRG.
	Sparse matrix method selection. This keyword may also be set with the "sys126" command line keyword. See the MSC 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 for 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 or bytes followed by one of the modifiers: "T", "TW", "TB", "G", "GW", "GB", "M", "MW", "MB", "K", "KW", "KB", "W", "B". See Specifying Memory Sizes for a description of these modifiers.		
Note:	No attempt is made to verify if the SCRATCH DBset can ever grow to the size specified by this keyword.		
	Example:	*nast_ver* `example sscr=1024gb`	
	Defines the default size of the SCRATCH DBset as 1 TB.		
	Example:	*nast_ver* `example sscr=.5tb`	
	Defines the default size of the SCRATCH DBset as .5TB or 512GB.		
submit (LINUX)	submit=[*list*=]*definition*	Default:	None
	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 in ASSIGN and INCLUDE statements and in command line arguments. This keyword may be specified in initialization or RC files and on the command line. The symbol definition may include references to previously defined symbols or environment variables using the standard "$*name*" or "${*name*}" syntax on LINUX or %*name*% syntax on Windows. For convenience, the character separating the "symbol" and "*name*" specification and the "*name*" and "*string*" specification may be either an equal sign ("='") or a hash mark ("#"). The use of a hash mark allows this keyword to be specified as an argument to a Windows .bat file.		
	If "node" is specified, symbolic names defined using this keyword are not used on the local system. Instead the specified values are passed to the remote system. This means that any pathnames must be valid on the remote system. Use the "lsymbol" keyword to specify symbolic names for the local system.		
	If "node" is not specified, symbolic names defined using the "lsymbol" keyword are processed as if they were defined using the "symbol" keyword.		
	Symbolic names are processed in the order they are encountered while processing the initialization and RC files and the command line. If a duplicate symbolic name is encountered, the new value replaces the previously specified value.		
	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 an ASSIGN or INCLUDE statement 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 (or "lsymbol=" keyword if "node" was not specified) in an initialization or RC file, on the command line, or as 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		
	`export DBSDIR=/dbs`		
	at a Korn shell prompt, or		
	`setenv DBSDIR /dbs`		
	at a C-shell prompt, or		
	`set DBSDIR=d:\dbs`		
	at a Windows shell prompt, is equivalent to the "symbol" keyword definition.		
Note:	If a symbolic name is defined by both a symbol statement in an RC file and by 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 interest 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=*option,option,...*	Default:	None
	Defines a global SYS value that is applied to Dbsets. Each option must have one of the following formats: *keyword=value* or *LNAMEXP*(*keyword=value,keyword=value,...*)		
where:			
	LNAMEXP	=	specifies a logical name expression using the LINUX/Windows file name specification format
	Characters may be specified in any case. Internally, they are converted to upper-case before they are used.		
	Most characters in a substitution pattern match themselves but you can also use some special *pattern-matching characters* in the pattern.		
	These special characters are:		

	*	=	Matches any string, including the null string.
	?	=	Matches any one character.

[...]	=	Matches any *one* of the characters enclosed in the square brackets.
[!...]	=	Matches any one character *other than* one of the characters that follow the exclamation mark within square brackets. Inside square brackets, a pair of characters separated by a - (minus) specifies a set of all characters that collate within the range of that pair, as defined by the ASCII collating sequence, so that [a-dy] is equivalent to [abcdy].
keyword=value	=	Specifies a keyword and value to be used for the Dbset file. If the entry is part of an option qualified by an LNAMEXP expression, the keyword and value will only be used for a Dbset file whose logical name is selected by the expression specified by LNAMEXP. Otherwise, the keyword and value will be used for all Dbset files. Note that a null LNAMEXP expression will match any logical name.

	The "sysfield" keyword may be specified more than once. The options are processed in the order specified on the various specifications. If multiple "keyword=value" options specify the same keyword, the last one encountered is the one that is used. You may use the "whence" keyword to see the "sysfield" keyword values. Also, the "sysfield" keywords are listed in the LOG file. See the sections titled Using the SYS Field or SYS Parameter Keywords for details on the valid keyword options.	
	Example:	*nast_ver* example sysfield=lock=no
	This example disables file locking for all Dbsets.	
	Example:	*nast_ver* example sysfield=lock=no sysfield=scr*(mapio=yes,lock=yes)
	This example disables file locking for all files and then enables filemapping ("mapio=yes") and turns file locking back on for Dbsets whose logical names start with "SCR". The end result is that file locking is disabled for all Dbsets except those whose logical names start with "SCR" and file mapping and file locking are enabled for Dbsets whose logical names start with "SCR".	

sys*n*	sys*n*=*value*	Default:	None
	Sets the SYSTEM(*n*) cell 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. If there is a "name" associated with the SYSTEM(*n*) value, that keyword will also be set to *value*. The System Cell number to System Cell name equivalence is listed in the Executing MSC Nastran in the *MSC Nastran Quick Reference Guide*. The form "system(*n*)=*value*" may be used, but the entire keyword-value string must be quoted when used on a LINUX command line.		
	Example:	*nast_ver* `example sys114=200`	
	or	*nast_ver* `example "system(114)=200"`	
	These examples set SYSTEM(114) to 200 GINO blocks. The second example shows how to quote the parenthetic form. Also, in this example, since SYSTEM(114) has the name "BUFFPOOL", the value of the "buffpool" keyword is also set to 200 GINO blocks.		
t3skew	t3skew=*number*	Default:	30.0
	Controls minimum vertex angle for TRIA3 elements at which User Warning Message 5491 is issued. See the MSC Nastran Quick Reference Guide, Section 1, The NASTRAN Statement, for more information on this keyword.		
tetraar	tetraar=*number*	Default:	100.0
	Specifies maximum allowable aspect ratio of longest to shortest edge for the CTETRA element. See the Executing MSC Nastran in the *MSC Nastran Quick Reference Guide* for more information on this keyword.		
uds	Path to User Defined Subroutine name	Default:	none
	User defined subroutine source file name. Refer to the *MSC Nastran User Defined Services User's Guide* for more details.		
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:	*nast_ver* `example node=othernode user=fred`	
	This example will run MSC 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 MSC 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:	*nast_ver* `example version=2023.1`	
	This example will run MSC Nastran V2023.1 assuming it has been installed in the same installation base directory as this version of MSC Nastran.		
whence	whence=*keyword_list*	Default:	None
	Displays value and source for listed keywords. This keyword may be used to determine a keyword's value and where it was set. An input datafile (JID) is optional; the job will not be run. If the "node" keyword is specified, the request will be passed to the remote node for processing. Otherwise, information will be displayed for the local node. If multiple "whence" keywords are specified, the keywords in the various keyword lists will be concatenated, except that if a null list is specified, all existing keywords in the accumulated list will be deleted. Any keywords in the "keyword_list" that have the format "sys*n*" will attempt to return the value associated with the System Cell name associated with system cell *n*, if possible. The entries in the "keyword_list" may also request information about a PARAM name. These entries have the format "`p:name`", where "`name`" is the PARAM name (*not* the name of the associated PARAM keyword, if any).		
	Normally, the output is two lines for each keyword. The first line specifies the "source", i.e., from where the keyword value is obtained; the second line specifies the keyword and its value. The only exception is when the keyword is "symbol", "system" or "j.params". In these cases, there will be multiple lines of keyword value information. If an unknown keyword is specified, a "User Warning Message" will be generated and the keyword will be ignored.		

	`Example:`	*nast_ver* `iter=yes`
		`whence=sys1,bpool`
		`whence=sscr,iter`	

	Assuming that none of these values is modified in configuration files, the output from this request is:		
	`MSC Nastran V2023.1 (...) ...`		
	`$ internal default`		
	`  sys1=8193`		
	`$ internal default`		
	`  bpool=37`		
	`$ internal default`		
	`  sscr=250000`		
	`$ command line[1]`		

	iter=yes
xdbold	xdbold={yes|no} Default=yes Keeps or deletes the previous copy of the .xdb. If "yes" is specified, the previous copy is kept and appended by the current run. If "no" is specified, the previous copy is deleted and a new file is created. This gives the same behavior as below FMS statement: ASSIGN dbc=<file_name>.xdb delete Example: nast20231 example xdbold=no

SYS Parameter Keywords

The following keywords may be used for DBset files and for DBC Fortran files.

async (See Table 4-7)	async=yes,no,must	Default:	No
	This keyword specifies the file is to be read using asynchronous I/O. If "async=yes" is specified and a memory allocation operation fails, then unbuffered disk I/O will be used. If "async=must" is specified and a memory allocation operation fails, then a fatal error will be issued and the job terminated. See Using Asynchronous I/O for further information.		
buffio (See Table 4-7)	buffio=yes,no,must	Default:	No
	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 MSC 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 (See Table 4-7)	mapio=yes,no,must	Default:	No
	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.		

report	report=yes,no	Default:	No
	Requests that a summary report about the number of file operations and other information about the I/O processing done for a particular file be written to the file defined by stderr when the file is closed. In addition, if TIMING=YES is specified, this report will contiain timing information about the various steps involved in the I/O processing. If ASYNC=YES, BUFFIO=YES or MAPIO=YES, the report will contain additional information about the processing specific to these methods.		
timing	timing=yes,no	Default:	No
	Requests that operation timing be enabled for the file. This timing information will be included in the .f04 file and, if REPORT=YES is also in effect, in the report written to stderr.		
wnum	wnum=*number*	Default:	4 (ASYNC=NO) 8 (ASYNC=YES)
(See Table 4-7)	Specifies the number of windows or buffers that will be maintained for each mapped, buffered or asynchronous I/O 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 32 inclusive, values outside of this range are ignored without acknowledgement.		
wsize	wsize=*size*	Default:	See text.
(See Table 4-7)	**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. The default is 128KB or 4*BUFFSIZE, whichever is larger.		
(See Table 4-7)	**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 default is 4*BUFFSIZE or 64K, whichever is larger.		

(See Table 4-7)	**Asynchronous I/O.** Specifies the size of the buffer used to hold data read from 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 default is 8*BUFFSIZE or 64KB, whichever is larger.
	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 the address space limit or physical memory cannot be determined for a particular platform, a value of 8GB for 64-bit pointer systems is used as the 25% limit value. 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 25% limit can be overridden if the numeric value is specified as a negative number. The 25% test will be suppressed and the actual window size value will be the absolute value of the specified numeric value. It is the user's responsibility to ensure that the specified value is valid and does not cause performance problems.) 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.

Environment Variables

The following environment variables affect the execution of the nastran command.

Table 2-1 Environment Variables Affecting the nastran Command

Name	Purpose
DISPLAY	LINUX: The default display for xmonast.
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.
MSC_ARCH	Specifies the MSC Nastran architecture.
MSC_BASE	If set, the script will use this directory as the *install_dir*.
MSC_ISHELLEXT	Alternate means to set the "ishellext" keyword.
MSC_ISHELLPATH	Alternate means to set the "ishellpath" keyword.

Table 2-1 Environment Variables Affecting the nastran Command (continued)

Name	Purpose
MSC_JIDPATH	Alternate means to set the “jidpath” keyword.
MSC_LICENSE_FILE	Alternate means to set the “authorize” keyword.
MSC_NOEXE	If set, the nastran command will build the execution script but will not actually execute it. This may be useful for debugging purposes.
MSC_OLDTYPES	Alternate means to set the “oldtypes” keyword.
MSC_VERSD	MSC use only.
MSCDBG	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 2-2 “Pre” and “Post” Keyword Environment Variables

Name	Purpose
DBSDIR	The directory part of MSC_DBS, i.e., the directory that will contain the permanent database files.
DELDIR	Directory containing the solution sequence source files (*install_dir*/*prod_ver*/nast/del on LINUX and *install_dir**prod_ver*\nast/del on Windows).
DEMODIR	Directory containing DEMO library (*install_dir*/*prod_ver*/nast/demo on LINUX and *install_dir**prod_ver*\nast\demo on Windows).
JIDDIR	Directory containing the input file.
MSC_APP	yes,no
MSC_ASG	MSC use only.
MSC_ARCH	The actual architecture used by the nastran command.
MSC_LICENSE_FILE	Licensing value.
MSC_BASE	The actual *install_dir* used by the nastran command.
MSC_DBS	Default prefix of permanent databases.

Table 2-2 "Pre" and "Post" Keyword Environment Variables (continued)

Name	Purpose
MSC_EXE	Executable path.
MSC_JID	Input data file path.
MSC_MEM	Open core memory size in words.
MSC_OLD	yes,no
MSC_OUT	Prefix of F06, F04, and LOG files.
MSC_SCR	yes,no
MSC_SDIR	Default prefix of scratch databases.
MSC_VERSD	MSC use only.
OUTDIR	Output file directory.
SSSALTERDIR	Directory containing SSS alters (*install_dir*/*prod_ver*/nast/sssalter on LINUX and *install_dir**prod_ver*\nast\misc\sssalter on Windows).
TEMP	Windows: Temporary directory.
TMPDIR	LINUX: Temporary directory.
TPLDIR	Directory containing TPL library (*doc_install_dir*/tpl (or tpl6) on LINUX and *doc_install_dir*\tpl (or tpl6) on Windows).

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 2-3 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.kwds=*filename*	Pathname of User-defined general keywords file
0.lcl	Local job template pathname. The path needs to be fully qualified.
0.params=*filename*	Pathname of User-defined PARAM keywords file

Table 2-3 Other Keywords (continued)

Keyword	Purpose
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.addall=*list*	Comma separated list of extensions to be added to the j.all list
a.addapp=*list*	Comma separated list of extensions to be added to the j.app list.
a.addofp=*list*	Comma separated list of extensions to be added to the j.ofp list.
a.addold=*list*	Comma separated list of extensions to be added to the j.old list.
a.appdir	Application specific base pathname relative to MSC_BASE.
a.altmode	The INTEGER mode associated with the alternate architecture.
a.altmodedir	The directory name associated with the alternate architecture.
a.archdir	Architecture specific base pathname relative to MSC_BASE.
a.estimate	ESTIMATE executable filename relative to “a.archdir”.
a.exedir	Directory part of any file name specified by “executable”.
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	Version dependent RC files base filename. For LINUX, default is “nast<*vernum*>rc” and for Windows, default is “nast<*vernum*>.rcf”.
a.rcb	Version independent RC base filename. Default is the “a.rc” keyword value.
a.release	Release number, same as MSC 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	MSC internal variable.
a.touch	News file touch pathname.
a.urc	Version dependent User and Local RC files base filename. For LINUX, will always be prefixed by “.”. For LINUX, default is “nastranrc” and for Windows default is “nastran.rcf”.

Table 2-3 Other Keywords (continued)

Keyword	Purpose
a.urcb	Version independent User and Local RC files base file name. Default is the “a.rcb” keyword value. For LINUX, will always be prefixed by “.”.
d.dbsds	Blank separated list of per-task “dbs” directory values of DMP jobs.
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.
datecmd	Date command. Only used on Windows.
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.argv	Comma separated list of keywords and values to be added to the r.argv argument list.
j.base	Job basename.
j.command	Job submittal command string.
j.dir	Job directory.
j.env	Job environment variable list.
j.expbase	Generated <expjid> base name.
j.expdir	Generated <expjid> directory.
j.expjid	Generated <expjid> file name.
j.ext_xdb	File type associated with logical name DBC files. Built from "dbc(xdb)" as modified using "extdefault".
j.extall	Blank separated list of output file types (extensions) to be deleted at job completion if "delete-all" is specified. Built from "j.all" and "a.addall" as modified using "extdefault".
j.extapp	Blank-separated list of output file types (extensions) to be appended at job completion if "append=yes" is specified. Built from "j.app" and "a.addapp" as modified using "extdefault".

Table 2-3 Other Keywords (continued)

Keyword	Purpose
j.extofp	Blank-separated list of output file types that will be deleted at job completion if and only if they are empty. Built from "j.ofp" and "a.addofp" as modified using "extdefault".
j.extofp_old	Blank-separated list of output file types that will be used in "dmparallel" mode to define the files to be merged or deleted. (See "mergeresults".) Built from "j.ofp", "a.addofp" and "oldtypes" as modified using "extdefault".
j.extold	Blank-separated list of output file types that will be versioned ("old=yes") or deleted ("old=no") at job start. Built from "j.old", "a.addold" and "oldtypes" as modified using "extdefault".
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.params	Generated list of PARAM statements. Contains the result of INI file, RC file and command line PARAM processing
j.rcfiles	Comma-separated list of RC files.
j.server	MSC 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.

Table 2-3 Other Keywords (continued)

Keyword	Purpose
r.altmode	The INTEGER mode associated with the remote mode alternate architecture.
r.altmodedir	The directory name associated with the remote node alternate architecture.
r.argv	List of arguments to be processed on rmt/dmp host.
r.expvis	"expjid" visibility flag for remote job. Value is "yes" or "no".
r.jidvis	JID visibility flag.
r.oscode	Remote system operating system code, 1 = Windows, 2 = LINUX.
r.outvis	Output directory visbility flag.
r.rmtcode	Remote communications protocol, 1 = mscrmtcmd, 2 = ssh/scp.
r.rshell	Remote node Shell pathname. Only used when "r.rmtcode" is 1.
s.arch	System architecture name.
s.block	Words per disk block.
s.bpw	Bytes per word.
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 LINUX and Windows.
s.proc	Default processor subtype.
s.rawid	Raw configuration number.
s.scp	Remote file copy command.
s.ssh	Remote shell command.
s.type	System description.
s.vmem	Virtual memory, in MB. Only known on Windows.
tcmd	Timing command.

System Cell Keyword Mapping

The following table lists the System Cell Name - System Cell Number equivalence used by MSC Nastran when processing the sys*n* and whence keywords:

Table 2-4 System Cell Name -- System Cell Number

System Cell Name	System Cell Number
attdel	124
autoasgn	133
bfgs	145
buffpool	114
buffsize	1
ifpbuff	624
chexaint	212
config	28
cordm	204
cpyinput	305
dblamkd	155
dbverchk	148
diaga	25
diagb	61
disksave	193
distort	213
f04	86
f06	2
fastio	194
fbsmem	146
fbsopt	70
frqseq	195
hicore	57
iter	216
ldqrkd	170
locbulk	143
massbuff	199
maxset	263

Table 2-4 System Cell Name -- System Cell Number

System Cell Name	System Cell Number
metime	20
mindef	303
minfront	198
mperturb	304
mpyad	66
newhess	108
nlines	9
nsegadd	200
numseg	197
punch	64
q4skew	190
q4taper	189
quadint	141
radlst	88
radmtx	87
real	81
scr300	142
scr300del	150
scrsave	196
smp	107
sparse	126
t3skew	218
tetraar	191
usparse	209

C System Descriptions

- Overview
- System Descriptions
- Numerical Data
- Computer Dependent Defaults

Overview

This appendix presents quantitative information for evaluating the processing requirements of MSC Nastran. It includes system descriptions, numerical data, and information on computer dependent defaults.

Binary File Byte Ordering (Endian)

The term "endian" refers to the byte ordering for numeric data used by a particular computer architecture. "Big-endian" specifies that the most significant byte (MSB) of a data element is stored at the lowest byte address, while "little-endian" specifies that the least significant byte (LSB) of a data element is stored at the lowest byte address. Most LINUX platforms are big-endian machines, while all Intel x86 and compatible platforms, e.g., Intel Xeon, including those running both Windows and Linux, are little-endian machines. Some architectures can be run in either endian mode. For example, the Intel Itanium processor runs in big-endian mode when running HP-UX and in little-endian mode when running Linux or Windows.

System Descriptions

Table 3-1 System Description – Intel – Linux

Item	Description
Supported Model(s)	Intel and Intel-compatible
Operating System(s)	RedHat 7.9, 8.6, SuSE 12SP5, 15SP3
Compiler	
Linux 64	Intel oneAPI 2022.1
Compiler Options	
FORTRAN	
Linux 64	-nbs -cm -pad_source -save -zero -w90 -WB -W0 -Qdyncom XNSTRN -mp1 -pc80 -O2 -fPIC
C++	
Linux 64	-O2 -shared -Bsymbolic
Word Length	64 bits
Build Type	LX8664 ILP64
Memory Management	Virtual

Table 3-2 System Description – Intel – Windows

Item	Description
Supported Model(s)	Intel and Intel-compatible
Operating System(s)	Windows 10 (21H2, 22H2). Windows 22 (22h2)
Compiler	
Windows 64	Intel oneAPI 2022.1 MSVC 14.31 Visual Studio 2022 17.1.7 Microsoft .NET 6
Compiler Options	
FORTRAN	
Windows 64	/fixed /nologo /nbs /WB /MD /Quppercase /Qdyncom XNSTRN /Qzero /Qsave /O2 /cm /IC:\Program
C++	
Windows 64	/MD /EHs /O2 /W3 /Wp64 /FC /nologo
Word Length	64 bits
Build Type	X8664 ILP64
Memory Management	Virtual

Numerical Data

Table 3-3 Numerical Data – 64-bit, little endian

Item	Description					
INTEGER Bit Representation	0	1				63
	S	Integer				
REAL Bit Representation	0	1	15	16		63
	S	Exponent		Mantiss		
Exponent Range for a REAL Number	±2644					
Precision of a REAL Variable	14 digits (48 bits)					

Table 3-3 Numerical Data – 64-bit, little endian

Item	Description					
DOUBLE PRECISION Bit Representation	0	1	15	16		63
	S	Exponent		Mantissa		
	64	79	80			127
	(Unused)		Mantissa (continued)			
Exponent Range for a DOUBLE PRECISION Number	±2466					
Precision of a DOUBLE PRECISION Variable	28 digits (96 bits)					

Computer Dependent Defaults

These tables list the computer-dependent default values for MSC Nastran. The default rank values are listed in Table 3-5.

Table 3-4 Computer-Dependent Defaults,

Parameter	Input File Settings	Command Line Settings	Default	Comment
BUFFPOOL	`NASTRAN BUFFPOOL=n`	`bpool=n`	150	GINO Blocks
BUFFSIZE	`NASTRAN BUFFSIZE=n`	`buffsize=n`	32769	Max: 65537
BUFFSIZE Increment	`NASTRAN SYSTEM(136)=n`	`sys136=n`	128	Words
DBALL Size	`INIT DBALL , LOGICAL=(DBALL(n))`	`sdball=n`	50,000	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

Table 3-4 Computer-Dependent Defaults,

Parameter	Input File Settings	Command Line Settings	Default	Comment
MAXSET	`NASTRAN MAXSET=n`	`sys263=n`	7	
SCRATCH Size	`INIT SCRATCH , LOGICAL=(logname(n)), SCR300=(logname(n))`	`sscr=n`	10,000,000	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 3-5 Computer-Dependent Default Rank Values

Computer Type	Model	SYS198	SYS205
Linux 64 Xeon	All	27	64
Linux 64 Opteron	All	6	64
Windows (64 bit)	All	8	8

Glossary

3060	A User Fatal Message indicating that authorization to run MSC Nastran has been denied (see Managing MSC Nastran Licensing, 34).
6080	A User Warning Message indicating that timing blocks must be generated for your computer (see Generating a Timing Block for a New Computer, 49).
acct	MSC accounting file directory, "*install_dir*/acct" on LINUX and "*install_dir*/acct" on Windows. Also, the program (*install_dir*/*prod_ver*/*arch*/acct on LINUX and *install_dir**prod_ver**arch*\acct.exe on Windows) that updates the current month's accounting data file. See MSCACT for the program source.
architecture RC files	The RC files residing in the "*install_dir*/conf/*arch*" directory on LINUX and in the "*install_dir*\conf*arch*" directory on Windows. See Command Initialization and Runtime Configuration Files, 138 in Appendix A for information about the names of these files and 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 MSC 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.
BUFFSIZE	One plus the number of words in a GINO physical record. Also, the NASTRAN statement keyword that sets the default buffer size.
conf	The MSC 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 MSC 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.
dbl700	Specifies that SOL 700 will use double precision. This keyword is only used if PATH=3 is specified on the SOL 700 entry and no sol700.pth file exists. The default is dbl700=no. Note that significant performance degradation will occur if "dbl700=yes" is specified.

DBset	Database file set.
DDLPRT	Utility program that prints the contents of the results database (XDB) data definition language database (*install_dir*/*prod_ver*/*arch*/dbc.xdb on LINUX and *install_dir**prod_ver**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*/*prod_ver*/*arch*/dbc.xdb on LINUX and *install_dir**prod_ver**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*/*prod_ver*/nast/demo on LINUX and *install_dir**prod_ver*\nast\demo on Windows) contains a selection of MSC Nastran input files that are documented in the MSC 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 MSC Nastran solution sequences.
DMP	Distributed Memory Parallel. In MSC Nastran, DMP execution is enabled by the “dmparallel” keyword.
dmp700	Specifies the number of hosts for an SOL 700 run. This keyword is only used if PATH=3 is specified on the SOL 700 entry and no sol700.pth file exists. The default hosts are the same for MSC Nastran.
doc	Documentation file directory.
ESTIMATE	Utility that estimates memory and disk requirement of a data file and make suggestions on improving the performance of MSC Nastran.
F04	The F04 file is created by MSC 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 MSC Nastran and contains the numerical results of the analysis. The F06 file has the file type “.f06".
file locking	A mechanism to prevent multiple MSC Nastran 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. MSC Nastran can use file mapping to access GINO files. See Table 4-7 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.
GINO	The MSC Nastran database subsystem.
GINO block	A block of data transferred by GINO.
IEEE	Institute of Electrical and Electronics Engineers, Inc. A professional society. The floating point formats and, to a lesser extent, algorithms used on most MSC Nastran computers are defined by IEEE Standard 754.
IFPBUFF	Specifies the physical record size, in words, of MSC Nastran IFPStar database. The physical I/O size is IFPBUFF-1 words. The maximum value of IFPBUFF is 65537 words.
INCLUDE	A general MSC 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.
local RC files	The RC files residing in the directory containing the input data file. See Command Initialization and Runtime Configuration Files, 138 in Appendix A for information about the names of these files.
LOG	The LOG file is created by MSC 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.
mem700	The default memory (in MegaWords) used the dynamic portion of SOL 700 runs. This keyword is only used if PATH=3 is specified on the SOL 700 entry and no sol700.pth file exists.
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.
MSC.ACCESS	FORTRAN-callable subroutine library that reads and writes results database (XDB) files.
MSCACT	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*/*prod_ver*/util/mscact.c on LINUX and *install_dir**prod_ver*\util\mscact.c on Windows).
MSGCMP	Utility program that compiles a text file to create a message catalog.
NAO	The Network Authorization Option of MSC 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 files	The RC files residing in the "*install_dir*/conf/net/*nodename*" directory on Linux and "*install_dir*\conf\net*nodename*" directory on Windows. See Command Initialization and Runtime Configuration Files, 138 in Appendix A for information about the names of these files.
NUSR	The node-locked license enforcement of the maximum number of users concurrently running MSC Nastran. See Enabling Account ID Validation, 38 for additional information.
on2	Default neutral-format OUTPUT2 file type.
op2	Default binary-format OUTPUT2 file type.
open core	Amount of working memory in words.
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 MSC Nastran to control execution parameters.
RCOUT2	Utility program that converts a neutral OUTPUT2 (.on2) file to a binary OUTPUT2 (.op2) file.
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 MSC Nastran.
SMEM	Scratch memory area for memory-resident database files.
smemory	Command line keyword to set SMEM.
SMP	Shared Memory Parallel. In MSC 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*/*prod_ver*/*arch*" on LINUX and "*install_dir**prod_ver**arch*" on Windows; the source files are found in "*install_dir*/*prod_ver*/nast/del" on LINUX and "*install_dir**prod_ver*\nast/del" on Windows.

SSSALTER	Additional alter and error corrections library, "*install_dir*/*prod_ver*/nast/sssalter" on LINUX and "*install_dir**prod_ver*\nast\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 files	The RC files residing in the "*install_dir*/conf" directory on LINUX and in the "*install_dir*\conf" directory on Windows. See Command Initialization and Runtime Configuration Files, 138 in Appendix A for information about the names of these files.
SYSTEM(x)	System cells that are used by MSC Nastran to control analysis parameters.
TABTST	Sample program that reads binary-format OUTPUT2 files.
TPL	The test problem library (TPL, *doc_install_dir*/tpl (or tpl6) on LINUX and *doc_install_dir*\tpl (or tpl6) on Windows) contains a general selection of MSC Nastran input files showing examples of most of the MSC Nastran capabilities, in general, these files are not documented. These files may be installed from the Documentation/Test File Installer.
TPLDIR	The tpl or tpl6 directories that are available in documentation/ example problem installer.
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 MSC Nastran has been denied (see Using the "mscinfo" Command (LINUX), 34).
UIM	A User Information Message that provides general information.
uncounted license	An uncounted license is a FLEXlm license that allows any number of concurrent executions of MSC Nastran on a given node. An uncounted license does not require a FLEXlm license server.
user RC files	The RC files residing in the "$HOME" directory on LINUX and in the "%HOMEDRIVE%%HOMEPATH%" directory on Windows. See Command Initialization and Runtime Configuration Files, 138 in Appendix A for information about the names of these files.
util	Utility program library, "*install_dir*/*prod_ver*/util" on LINUX and "*install_dir*\/*prod_ver*\util" on Windows.
UWM	A User Warning Message that warns of atypical situations. You must determine whether a problem exists in the analysis.

UWM 6080	A User Warning Message indicating that timing blocks must be generated for your computer (see Generating a Timing Block for a New Computer, 49).
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 MSC Nastran and contains results information for use by various post-processing programs. See the "POST" parameter in Parameters (p. 793) in the *MSC Nastran Quick Reference Guide* for further information on generating XDB files. XDB files are not versioned. The XDB file has the file type ".xdb".