# Table of Contents

1. **Introducing APDL** ........................................................................................................... 1  
2. **Working with the Toolbar** ................................................................................................. 3  
   2.1. Adding Commands to the Toolbar .................................................................................. 3  
   2.2. Modifying the Toolbar ..................................................................................................... 3  
      2.2.1. Example: Adding a Toolbar Button ......................................................................... 4  
      2.2.2. Saving Toolbar Buttons ......................................................................................... 5  
   2.3. Nesting Toolbar Abbreviations ....................................................................................... 5  
3. **Using Parameters** ............................................................................................................. 7  
   3.1. Guidelines for Parameter Names .................................................................................... 8  
      3.1.1. Hiding Parameters from *STATUS ......................................................................... 8  
   3.2. Defining Parameters ....................................................................................................... 9  
      3.2.1. Assigning Parameter Values During Execution .................................................... 9  
      3.2.2. Assigning Parameter Values At Startup .................................................................. 9  
      3.2.3. Assigning ANSYS-Supplied Values to Parameters ............................................ 10  
         3.2.3.1. Using the *GET Command .............................................................................. 10  
         3.2.3.2. Using In-line Get Functions ............................................................................ 10  
      3.2.4. Listing Parameters ................................................................................................. 11  
   3.3. Deleting Parameters ....................................................................................................... 12  
   3.4. Using Character Parameters ......................................................................................... 12  
   3.5. Substitution of Numeric Parametric Values ................................................................. 13  
      3.5.1. Preventing Substitution .......................................................................................... 13  
      3.5.2. Substitution of Character Parametric Values ...................................................... 13  
         3.5.2.1. Forced Substitution ......................................................................................... 13  
         3.5.2.2. Other Places Where Character Parameters Are Valid ................................ 14  
         3.5.2.3. Character Parameter Restrictions ................................................................ 15  
   3.6. Dynamic Substitution of Numeric or Character Parameters ....................................... 15  
   3.7. Parametric Expressions ................................................................................................. 15  
   3.8. Parametric Functions .................................................................................................... 16  
   3.9. Saving, Resuming, and Writing Parameters ................................................................ 17  
   3.10. Array Parameters ......................................................................................................... 18  
      3.10.1. Array Parameter Basics ....................................................................................... 19  
      3.10.2. Array Parameter Examples .................................................................................. 20  
      3.10.3. TABLE Type Array Parameters .......................................................................... 21  
      3.10.4. Defining and Listing Array Parameters ............................................................. 22  
      3.10.5. Specifying Array Element Values ....................................................................... 23  
         3.10.5.1. Specifying Individual Array Values ............................................................. 23  
         3.10.5.2. Filling Array Vectors ...................................................................................... 25  
         3.10.5.3. Interactively Editing Arrays ............................................................................ 25  
         3.10.5.4. Filling an Array From a Data File Using *VREAD ........................................ 26  
         3.10.5.5. Filling a TABLE Array From a Data File Using *TREAD ............................... 26  
         3.10.5.6. Interpolating Values ...................................................................................... 30  
         3.10.5.7. Retrieving Values into or Restoring Array Parameter Values ...................... 31  
         3.10.5.8. Listing Array Parameters .............................................................................. 32  
      3.10.6. Writing Data Files ................................................................................................. 33  
         3.10.6.1. Format Data Descriptors ............................................................................... 33  
      3.10.7. Operations Among Array Parameters .................................................................. 35  
         3.10.7.1. Vector Operations .......................................................................................... 35  
         3.10.7.2. Matrix Operations .......................................................................................... 37  
         3.10.7.3. Specification Commands for Vector and Matrix Operations ...................... 39  
      3.10.8. Plotting Array Parameter Vectors ...................................................................... 42
List of Tables

4.1. _RETURN Values .................................................................................................................. 59
B.1. *GET - Get Function Summary ......................................................................................... 81
Chapter 1: Introducing APDL

APDL stands for ANSYS Parametric Design Language, a scripting language that you can use to automate common tasks or even build your model in terms of parameters (variables). While all ANSYS commands can be used as part of the scripting language, the APDL commands discussed here are the true scripting commands and encompass a wide range of other features such as repeating a command, macros, if-then-else branching, do-loops, and scalar, vector and matrix operations.

While APDL is the foundation for sophisticated features such as design optimization and adaptive meshing, it also offers many conveniences that you can use in your day-to-day analyses. In this guide we'll introduce you to the basic features - parameters; macros; branching, looping, and repeating; and array parameters - and show you some simple examples. As you become more adept at the language, you will begin to recognize applications for APDL in your own environment.

This guide covers the following topics:

- **Working with the toolbar**: You can add frequently used ANSYS functions or macros to the ANSYS toolbar by defining abbreviations, which are aliases (up to eight characters long) for an ANSYS command, GUI function name, or macro name.
- **Using parameters**: Parameters are APDL variables (they are more similar to Fortran variables than to Fortran parameters). ANSYS uses two types of parameters: scalar and array.
- **Understanding APDL as a macro language**, including creating a macro: You can record a frequently used sequence of ANSYS commands in a macro file (these are sometimes called command files). Creating a macro enables you to, in effect, create your own custom ANSYS command. In addition to executing a series of ANSYS commands, a macro can call GUI functions or pass values into arguments.
- **Interfacing with the GUI**: Within an ANSYS macro, you have several ways to access components of the ANSYS GUI (toolbar, dialog box, *ASK command, etc.).
- **Encrypting macros**: ANSYS provides the ability to encrypt macro files so that the source is not "human-readable." Encrypted macros require an encryption key to run. You can either place the encryption key explicitly (in readable ASCII) in the macro or you can set it in ANSYS as a global encryption key.

See the APDL Commands for a list of all APDL commands discussed in this guide.
Chapter 2: Working with the Toolbar

You can add frequently used ANSYS functions or macros to the ANSYS toolbar by defining abbreviations, which are aliases (up to eight characters long) for an ANSYS command, GUI function name, or macro name. You can also modify the toolbar and nest toolbar abbreviations.

The following toolbar topics are available:

2.1. Adding Commands to the Toolbar
2.2. Modifying the Toolbar
2.3. Nesting Toolbar Abbreviations

2.1. Adding Commands to the Toolbar

You can add frequently used ANSYS functions or macros to the ANSYS toolbar (creating macros is covered starting in "APDL as a Macro Language"). You do this by defining abbreviations. An abbreviation is an alias (up to eight characters long) for an ANSYS command, GUI function name, or macro name. For example, MATPROP might be an abbreviation for a macro that lists material properties, SAVE_DB is an abbreviation for the SAVE command, and QUIT is an abbreviation for the Fnc(EXIT) function (which launches the Exit from ANSYS dialog box). APDL commands that can be used to define abbreviations are listed in Chapter 2 of the Command Reference.

The ANSYS program provides two ways to use abbreviations. You can issue the abbreviation (and execute the macro, command, etc. that it performs) by typing it at the beginning of a command line. If you are using the ANSYS GUI, you can also execute the macro or command by pressing the appropriate button on the ANSYS toolbar.

The toolbar shown in Figure 2.1: Toolbar (p. 3) contains buttons that correspond to existing abbreviations.

Figure 2.1: Toolbar

While some abbreviations, such as SAVE_DB, are predefined, the abbreviations the toolbar contains and the functions they execute are up to you. A single toolbar can hold up to 100 abbreviations (you can "nest" toolbars to extend this number). You can redefine or delete abbreviations at will; however, abbreviations are not automatically saved and must be explicitly saved to a file and reloaded for each ANSYS session.

2.2. Modifying the Toolbar

You can create abbreviations either through the *ABBR command or through the Utility Menu> Macro> Edit Abbreviations or Utility Menu> MenuCtrls> Edit Toolbar menu items. Using one of the menu items is preferable for two reasons:

• Clicking OK automatically updates the toolbar (using the *ABBR command requires that you use the Utility Menu> MenuCtrls> Update Toolbar menu item to make your new abbreviation appear on the toolbar).
You can easily edit the abbreviation if required.

The syntax for the *ABBWR command and related dialogs is

*ABBWR, Abbr, String

Abbr
The abbreviation name that will appear on the toolbar button. The name can contain up to eight characters.

String
The String argument is the name of the macro or command that Abbr represents. If String is the name of a macro, the macro must be within the macro search path. For more information about using macros, see "APDL as a Macro Language". If String references an ANSYS picking menu or dialog box (using UIDL), then specify "Fnc_string." For example, in the abbreviation definitions for "QUIT" and "POWRGRPH" shown above, "Fnc_/QUIT" and "Fnc_/GRAPHICS" are unique UIDL function names which identify the ANSYS picking menu or dialog box associated with the QUIT and POWRGRPH abbreviations respectively. For more information about accessing UIDL functions, see Calling Dialog Boxes From a Macro. String can contain up to 60 characters but cannot include any of the following:

- The character "$"
- The commands C***, /COM, /GOPR, /NOPR, /QUIT, /UI, or *END

The default ANSYS toolbar has the following abbreviations predefined:

*ABBWR, SAVE_DB, SAVE
*ABBWR, RESUM_DB, RESUME
*ABBWR, QUIT, Fnc_/EXIT
*ABBWR, POWRGRPH, Fnc_/GRAPHICS

2.2.1. Example: Adding a Toolbar Button

For example, to add a button to the toolbar that calls the macro file mymacro.mac, you would enter the values shown in the following figure in the Utility Menu> MenuCtrls> Edit Toolbar dialog box.

![Figure 2.2: Adding a New Abbreviation](image)

The new button is appended to the button bar as shown in the following figure.
2.2.2. Saving Toolbar Buttons

Toolbar buttons are not persistent from one ANSYS session to the next; however, they are saved and maintained in the database so that any "resume" of the session will still contain these abbreviations. To save your custom button definitions, you must explicitly save them to a file through the Utility Menu> MenuCtrls> Save Toolbar menu item (ABBSAV command) and restore them for each session using the Utility Menu> MenuCtrls> Restore Toolbar menu item (ABBRES command). You can do this programmatically in a macro.

Note

If any abbreviations already exist in the named file, the ABBSAV command overwrites them.

The format of the abbreviations file is the APDL commands that are used to create the abbreviations. Thus, if you wish to edit a large set of buttons or change their order, you may find using a text editor to be the most convenient method. For example, the following is the file that results from saving the default toolbar buttons.

```
/NOPR
*ABB,SAVE_DB,SAVE
*ABB,RESUM_DB,RESUME
*ABB,QUIT,Fnc_/EXIT
*ABB,POWRGRPH,Fnc_/GRAPHICS
/GO
```

The *ABB commands (the abbreviated form of *ABBR) define the buttons. The /NOPR at the top turns off echoing to the log file while the /GO at the bottom turns log file echoing on.

2.3. Nesting Toolbar Abbreviations

The save-and-restore features described above allow you to nest abbreviations. By nesting abbreviations under one button, you can define specialized toolbars (if you have many abbreviations, having them on a single toolbar can be cluttered, making it difficult to find the proper button). To nest abbreviations, you simply define an abbreviation that restores an abbreviation file. For example, the following command defines PREP_ABR as an abbreviation that restores abbreviations from the file prep.abbr.

```
*ABBR,PREP_ABR,ABBRES,,PREP,ABBR
```

PREP_ABR will appear as a button on the toolbar. Clicking it will replace the existing buttons with the set of buttons defined in the prep.abbr file.

By defining abbreviations to restore these files and including those abbreviations in the appropriate files, you can have a virtually unlimited number of abbreviations in a given ANSYS session. You can even extend this concept and create your own menu hierarchy by nesting several abbreviation files. If you implement such a hierarchy, it's a good practice to add an abbreviation as a "return" button in each file to navigate back through the menus.
Chapter 3: Using Parameters

Parameters are APDL variables (they are more similar to Fortran variables than to Fortran parameters). You don't need to explicitly declare the parameter type. All numeric values (whether integer or real) are stored as double-precision values. Parameters that are used but not defined are assigned a near-zero, or "tiny," value of approximately $2^{-100}$. For example, if parameter A is defined as $A=B$, and B is not defined, then A is assigned the tiny value.

ANSYS uses two types of parameters: scalar and array. The first part of this chapter discusses information that is applicable to both types. Starting with Array Parameters (p. 18), the information is specific to array type parameters. APDL commands used to define parameters in general and array parameters specifically are listed in Chapter 2 of the Command Reference.

Character strings (up to eight characters long) can be assigned to parameters by simply enclosing the string in single quotes. APDL also provides several types of array parameters: numeric, character, string and table (a special numeric type that automatically interpolates values).

You can use a parameter (instead of a literal number or character string) as an argument to any ANSYS command; the parameter is evaluated and its current value is used for that argument. For example, if you assign the value 2.7 to a parameter named AA and then issue the command

```
N, 12, AA, 4
```

the ANSYS program will interpret the command as

```
N, 12, 2.7, 4
```

(which defines node 12 at X=2.7 and Y=4).

Note

If array, table, or character parameters are used within a macro or input file, those parameters should be dimensioned (if array or table) and defined within that macro or input file. If you fail to follow this practice, ANSYS will produce error messages stating that those parameters are undefined. ANSYS will produce the error messages even if the parameters lie within unexecuted *IF statements, as parameter substitution is done before the branching for the *IF is checked.

The following parameter topics are available:

3.1. Guidelines for Parameter Names  
3.2. Defining Parameters  
3.3. Deleting Parameters  
3.4. Using Character Parameters  
3.5. Substitution of Numeric Parametric Values  
3.6. Dynamic Substitution of Numeric or Character Parameters  
3.7. Parametric Expressions  
3.8. Parametric Functions  
3.9. Saving, Resuming, and Writing Parameters  
3.10. Array Parameters
3.1. Guidelines for Parameter Names

Parameter names must:

• Begin with a letter
• Contain only letters, numbers, and underscore characters
• Contain no more than 32 characters

Examples of valid and invalid parameter names are

Valid:

  ABC
  PI
  X_OR_Y

Invalid:

  MY_PARAMETER_NAME_LONGER_THAN_32_CHARACTERS (more than 32 characters)
  2CF3 (begins with a number)
  M&E (invalid character "&")

When naming parameters:

• Avoid parameter names that match commonly used ANSYS labels, such as:
  – Degree of freedom (DOF) labels (TEMP, UX, PRES, etc.)
  – Convenience labels (ALL, PICK, STAT, etc.)
  – User-defined labels (such as those defined with the ETABLE command)
  – Array type field labels (such as CHAR, ARRAY, TABLE, etc.)
• Parameter names ARG1 through ARG9 and AR10 through AR99 are reserved for local parameters. Generally, local parameters are used in macros (see Local Variables). Use of these names as "regular" parameters is not recommended.
• Parameter names must not match abbreviations defined with the *ABBR command. For more information about abbreviations, see Adding Commands to the Toolbar.
• Do not begin parameter names with an underscore (_). This convention is reserved for parameters used by the GUI and ANSYS-supplied macros.
• APDL programmers supporting an organization should consider naming their parameters with a trailing underscore (_). These can displayed as a group using the *STATUS command and deleted from memory as a group through the *DEL command.
• Older ANSYS-supplied macro files may use parameter names that do not begin with an underscore. Using these macros embedded in your own macros may cause conflicts if the same parameter names are used.

3.1.1. Hiding Parameters from *STATUS

Listing Parameters (p. 11) discusses listing parameters through the *STATUS command. You can use a parameter naming convention to "hide" parameters from the *STATUS command. Any parameter whose name ends in an underscore (_) will not be listed by *STATUS.
This capability was added specifically for those who are developing APDL macros for large audiences. You can use this to build macros that your ANSYS users and other macro programmers cannot list.

3.2. Defining Parameters

Unless otherwise specified, the information in the next several sections applies to both scalar and array type parameters. Beginning with Array Parameters (p. 18), the information is specific to array type parameters.

You can either assign values to parameters or retrieve values supplied by ANSYS and store these values in parameters. For retrieving values from ANSYS, you can use either the *GET command or the various in-line get functions. The following sections cover these subjects in detail.

3.2.1. Assigning Parameter Values During Execution

You can use the *SET command to define parameters. The following examples illustrate a set of example parameters defined using *SET:

*SET, ABC, -24
*SET, QR, 2.07E11
*SET, XORY, ABC
*SET, CPARM, 'CASE1'

You can use an "=" as a shorthand way of calling the *SET command (this is the most convenient method). The format of the shortcut is Name = Value, where Name is the name assigned to the parameter and Value is the numeric or character value stored in that parameter. For character parameters, the assigned value must be enclosed in single quotes and cannot exceed eight alphanumeric characters. The following are examples of "=" in use:

ABC=-24
QR=2.07E11
XORY=ABC
CPARM='CASE1'

In the GUI, you can either type the "=" directly in the ANSYS input window or in the "Selection" field of the Scalar Parameter dialog box (accessed by the Utility Menu> Parameters> Scalar Parameters menu item).

3.2.2. Assigning Parameter Values At Startup

You can define parameters as arguments when launching ANSYS from the operating system command line. Simply type parameter definitions after the ANSYS execution command (which is system dependent) using the format -Name Value. For example, the following defines two parameters (parm1 and parm2) having the values 89.3 and -0.1:

ansys120 -parm1 89.3 -parm2 -0.1

It is a good practice to avoid assigning one or two character parameter names at startup to avoid conflicts with ANSYS command line options.

Note

Remember that UNIX shells treat single quotes and many other non-alphanumeric characters as special symbols. When defining character parameters, you must tell UNIX not to interpret the quotes by inserting a back slash (\) before the single quotes. For example, the following defines two character parameters having the values `filename' and `200. '

ansys120 -cparm1 `\'filename\'\' -cparm2 `\'200\'\''
If you use the ANSYS Launcher to start ANSYS, you can define parameters through the **Customization** tab (using the **Name Value** format described above).

If you are defining a large number of parameters at startup, you'll find it much more convenient to define these in the `start120.ans` file or through a separate file that you can load through the `/INPUT` command instead of the command line.

### 3.2.3. Assigning ANSYS-Supplied Values to Parameters

ANSYS provides two powerful methods for retrieving values:

- The **GET** command, which retrieves a value from a specified item and stores it in a specified parameter.
- The in-line get functions, which can be used in operations. Each get function returns a specific value from a specific item.

#### 3.2.3.1. Using the **GET** Command

The **GET** command (Utility Menu> Parameters> Get Scalar Data) retrieves an ANSYS-supplied value for an item (a node, an element, an area, etc.) and stores it as a user-named parameter. Various keyword, label, and number combinations identify the retrieved item. For example, **GET**, A,ELEM,5,CENT,X returns the centroid x-location of element 5 and stores the result as parameter A.

The format for the **GET** command is:

```
*GET,Par,Entity,ENTNUM,Item1,IT1NUM,Item2,IT2NUM
```

where

- **Par** is the name of the parameter to store the retrieved item.
- **Entity** is a keyword for the item to be stored. Valid keywords are NODE, ELEM, KP, LINE, AREA, VOLU, etc. For a complete list of valid keywords, see the **GET** description in the Command Reference.
- **ENTNUM** is the number of the entity (or zero for all entities).
- **Item1** is the name of an item for a particular entity. For example, if **Entity** is ELEM, **Item1** will be either NUM (the highest or lowest element number in the selected set) or COUNT (the number of elements in the set). (For a complete list of **Item1** values for each entity type, see the **GET** description in the Command Reference.)

You can think of the **GET** command as a path down a tree structure, from general to specific information.

The following examples show the **GET** command in use. The first command below gets the material attribute (the MAT reference number) of element 97 and assigns it to parameter BCD:

```
*GET,BCD,ELEM,97,ATTR,MAT     ! BCD = Material number of element 97
```

#### 3.2.3.2. Using In-line Get Functions

For some items, you can use in-line "get functions" in place of the **GET** command. A get function returns a value for an item and uses it directly in the current operation. This process allows you to bypass the dual

```
*GET,BCD,ELEM,97,ATTR,MAT
```

```
*GET,V37,ELEM,37,VOLU         ! V37 = volume of element 37
```

```
*GET,EL52,ELEM,52,HGEN        ! EL52 = value of heat generation in element 52
```

```
*GET,OPER,ELEM,102,HCOE,2     ! OPER = heat coefficient of element 102,face2
```

```
*GET,TMP,ELEM,16,TBULK,3      ! TMP = bulk temperature of element 16,face3
```

```
*GET,NMAX,NODE,,NUM,MAX      ! NMAX = maximum active node number
```

```
*GET,HNGD,NODE,12,HGEN        ! HNGD = value of heat generation at node 12
```

```
*GET,COORD,ACTIVE,,CSYS       ! COORD = active coordinate system number
```

### 3.2.3.3. Using ANSYS-Supplied Values in Operations

In addition to retrieving values, you can use ANSYS-supplied values in operations. For example:

```
*GET,BCD,ELEM,97,ATTR,MAT
*GET,HNGD,NODE,12,HGEN
```

```
*GET,OPER,ELEM,102,HCOE,2
```

```
*GET,TMP,ELEM,16,TBULK,3
```

```
*GET,NMAX,NODE,,NUM,MAX
```

```
*GET,COORD,ACTIVE,,CSYS
```

These commands assign the values retrieved from element 97, node 12, element 102, face 2, element 16, face 3, node number, and the active coordinate system, respectively.
steps of storing the value with a parameter name and then entering the parameter name in an operation. For example, suppose that you want to calculate the average x-location of two nodes. You could do the following using the *GET function:

1. Issue the following command to assign the x-location of Node 1 to parameter L1.

   *GET, L1, NODE, 1, LOC, X

2. Issue a second *GET command to assign the x-location of Node 2 to parameter L2.
3. Compute the middle location from \( \text{MID} = (L1 + L2) / 2 \).

A shorter method is to use the node location "get function" \( \text{NX}(N) \), which returns the x-location of node \( N \). You can use it to calculate the MID location without setting intermediate parameters L1 and L2, as is shown in the following example:

\[
\text{MID} = (\text{NX}(1) + \text{NX}(2)) / 2
\]

Get function arguments can themselves be parameters or other get functions. For instance, get function \( \text{NELEM}(\text{ENUM},NPOS) \) returns the node number in position NPOS for element ENUM. Combining functions \( \text{NX}(\text{NELEM}(\text{ENUM},NPOS)) \) returns the x-location of that node.

Get Function Summary summarizes the available get functions.

### 3.2.4. Listing Parameters

Once you have defined parameters, you can list them using the *STATUS command. If the *STATUS command is issued without arguments, it provides a list of all of the currently defined parameters. The following example shows the command and a typical listing.

```
*STATUS
PARAMETER STATUS- ( 5 PARAMETERS DEFINED)

NAME      VALUE         TYPE     DIMENSIONS
ABC       -24.0000000   SCALAR
HEIGHT    57.0000000    SCALAR
QR        2.070000000E+11 SCALAR
X.OR.Y    -24.0000000   SCALAR
CPARM     CASE1         CHARACTER
```

You can also access this information through either the Utility Menu> List> Other> Parameters or Utility Menu> List> Status> Parameters> All Parameters menu items.

---

**Note**

Any parameters beginning or ending in an underscore (_) are not shown by the *STATUS command.

You can check the status of individual parameters by providing these as arguments to the *STATUS command. The following example shows the status of the ABC parameter.

```
*STATUS, ABC
PARAMETER STATUS- abc ( 5 PARAMETERS DEFINED)

NAME      VALUE         TYPE     DIMENSIONS
ABC       -24.0000000   SCALAR
```

You can also check the status of specific parameters through the Utility Menu> List> Other> Named Parameter or Utility Menu> List> Status> Parameters> Named Parameters menu items.
3.3. Deleting Parameters

You can delete specific parameters in two ways:

- Issue the "=" command, leaving the right-hand side of the command blank. For example, to delete the QR parameter issue this command:

  QR=

- Issue the *SET command (Utility Menu> Parameters> Scalar Parameters), but don't specify a value for the parameter. For example, to delete the QR parameter via the *SET command issue the command as follows:

  *SET,QR,

Setting a numeric parameter equal to zero does not delete it. Similarly, setting a character parameter equal to empty single quotes (""") or placing blanks within single quotes does not delete the parameter.

3.4. Using Character Parameters

Typically, character parameters are used to provide file names and extensions. The desired file name can be assigned to a character parameter, and that parameter can be used anywhere a file name is required. Similarly, a file extension can be assigned to a character parameter and used where appropriate (typically the Ext command argument). In batch mode, this allows you to easily change file names for multiple runs by simply changing the initial alphanumeric "value" of the character parameter in your input file.

Note

Remember that character parameters are limited to a total of eight characters.

The following is a list of general uses for character parameters.

- As arguments to any applicable command field (that is, where alphanumeric input is expected)
- As macro name arguments for the *USE command (Utility Menu> Macro> Execute Data Block)

  NAME='MACRO'  ! MACRO is the name of a macro file
  *USE,NAME     ! Calls MACRO

- As arguments to macro calls for *USE and for the "unknown command" macro. Any of the following macro calls are allowed:

  ABC='SX'
  *USE,NAME,ABC

  or

  *USE,NAME,'SX'

  DEF='SY'
  NEWMACRO,DEF  ! Calls existing macro file NEWMACRO.MAC

  or

  NEWMACRO,'SY'
3.5. Substitution of Numeric Parametric Values

Whenever you use a parameter name in a numeric command field, its value is automatically substituted. If no value has been assigned to the parameter (that is, if the parameter has not been defined), a near-zero value \(2^{-100}\) will be substituted, usually without warning.

**Note**

Defining the parameter after it is used in a command does not "update" the command in most cases. (Exceptions are the commands /TITLE, /STITLE, *ABBR, and /TLABEL. See Forced Substitution (p. 13) for more information.) For example:

\[
\begin{align*}
Y &= 0 \\
X &= 2.7 \\
N, 1, X, Y & \quad \text{! Node 1 at } (2.7, 0) \\
Y &= 3.5 & \quad \text{! Redefining parameter } Y \text{ now does not update node 1}
\end{align*}
\]

3.5.1. Preventing Substitution

You can prevent parameter substitution by enclosing the parameter name with single quotes ('), for example, 'XYZ'. The literal string is then used; therefore, this feature is valid only in non-numerical fields.

Conversely, you can force parameter substitution in titles, subtitles, and filenames by enclosing the parameter name with percent signs (%). For example,

\[
/TITLE, \text{ TEMPERATURE CONTOURS AT TIME} = %TM%
\]

specifies a title in which the numerical value of parameter TM is substituted. Note that the parameter is substituted at the time the title is used.

3.5.2. Substitution of Character Parametric Values

Use of a character parameter in an alphanumeric command field generally results in automatic substitution of its value. Forced substitution and character parameter restrictions are explained below.

3.5.2.1. Forced Substitution

As with numerical parameters, you can force the substitution of a character parameter value in certain cases where substitution would not occur otherwise. This is done by enclosing the character parameter name with percent signs (%). Forced substitution of character parameters is valid for the following commands:

- **/TITLE** command (Title field). Specifies titles for various printed output.
- **/STITLE** command (Title field). Specifies subtitles, similar to /TITLE. (You cannot access the /STITLE command directly in the GUI.)
- **/TLABEL** command (Text field). Specifies text string for annotation.
- ***ABBR** command (Abbr field). Defines an abbreviation.

Forced substitution is also valid in the following types of fields:

- Any filename or extension command argument. These arguments apply to commands such as /FILNAME, RESUME, /INPUT, /OUTPUT, and FILE. (Direct parameter substitution is also valid in these fields.)
• Any 32 character field: A typical example is the name of macros. (Direct substitution is not valid for these fields.)

• As a command name in any command name field. Also as an "unknown command" macro name in field 1. For example:

```
R='RESUME'
%R,MODEL,DB
```

The following example of the command input method shows forced substitution for a subtitle definition and for a directory name.

```
A='TEST'
B='RST'
C='/ANSYS'
D='/MODELS/'
/STITLE,,RESULTS FROM FILE %C%%D%%A%%B%

SUBTITLE 1 =
RESULTS FROM FILE /ANSYS/MODELS/TEST.RST
```

3.5.2.2. Other Places Where Character Parameters Are Valid

In addition to the more general applications already discussed, there are some specific instances where character parameters are allowed for added convenience. The commands which are affected and details of usage are outlined below.

**ASK**

This command may prompt you for an alphanumeric string (up to eight characters enclosed in single quotes) which is assigned to a character scalar parameter. (You cannot access the *ASK command directly in the GUI.)

**CFWRITE**

This command writes ANSYS commands to the file opened by *CFOPEN. It can be used to write a character parameter assignment to that file. For example, *CFWRITE,B = 'FILE' is valid. (You cannot access the *CFWRITE and *CFOPEN commands directly in the GUI.)

**IF** and **ELSEIF**

Character parameters may be used for the VAL1 and VAL2 arguments of these commands. For the Oper argument, only labels EQ (equal) and NE (not equal) are valid when using character parameters. (You cannot access the *IF and *ELSEIF commands directly in the GUI.) Example:

```
CPARM='NO'
*IF,CPARM,NE,'YES',THEN
```

**MSG**

Character parameters are allowed as input for the VAL1 through VAL8 arguments. The data descriptor %C is used to indicate alphanumeric character data on the format line (which must follow the *MSG command). The %C corresponds to the FORTRAN descriptor A8. (You cannot access the *MSG command directly in the GUI.)

**PARSAV** and **PARRES**

These commands will save character parameters to a file (PARSAV command or menu path Utility Menu> Parameters> Save Parameters) and resume character parameters from a file (PARRES or Utility Menu> Parameters> Restore Parameters).
*VREAD
This command (Utility Menu> Parameters> Array Parameters> Read from File) can be used to read alphanumeric character data from a file and produce a character array parameter. The FORTRAN character descriptor (A) may be used in the format line which must follow the *VREAD command.

*VWRITE
This command (menu path Utility Menu> Parameters> Array Parameters> Write to File) can be used to write character parameter data to a file in a formatted sequence. The FORTRAN character descriptor (A) may be used in the format line which must follow the *VWRITE command.

3.5.2.3. Character Parameter Restrictions
Although character parameters have much of the same functionality as numerical parameters, there are several instances where character parameters are not valid.

- Character parameter substitution is not allowed for the Par argument of the *SET, *GET, *DIM, and *STATUS commands.
- Interactive editing of array parameters (*VEDIT command) is not available for character array parameters.
- Vector operation commands, such as *VOPER, *VSCFUN, *VFUN, *VFILL, *VGET, and *VITRP, do not work with character array parameters.
- When operating on character parameters, the specification commands *VMASK and *VLEN are applicable only to the *VWRITE and *VREAD commands.
- Character parameters are not valid in parametric expressions which use addition, subtraction, multiplication, etc.

3.6. Dynamic Substitution of Numeric or Character Parameters
Dynamic substitution of parameters will occur for the following commands: /TITLE, /STITLE, *ABBR, /AN3D, and /TLABEL. Dynamic substitution allows the revised value of a parameter to be used, even if the command which uses the parameter value has not been reissued.

Example:
XYZ='CASE 1'
/TITLE,This is %XYZ%
APLOT

The title "This is CASE 1" will appear on the area plot.

You can then change the value of XYZ and the new title will appear on subsequent plots, even though you did not reissue /TITLE.

XYZ='CASE 2'

The title "This is CASE 2" will appear on subsequent plots.

3.7. Parametric Expressions
Parametric expressions involve operations among parameters and numbers such as addition, subtraction, multiplication, and division. For example:

\[
\begin{align*}
X &= A + B \\
P &= \frac{(R2+R1)}{2} \\
D &= B + (E^2) - (4*A*C) \\
XYZ &= (A < B) + Y^2 \\
\end{align*}
\]

! Evaluates to D = \(-B + E^2 - 4AC\)

! Evaluates to XYZ = A + Y^2 if A is less than B;

! otherwise to XYZ = B + Y^2
The following is a complete list of APDL operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>**</td>
<td>Exponentiation</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less-Than Comparison</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater-Than Comparison</td>
</tr>
</tbody>
</table>

You can also use parentheses for clarity and for "nesting" of operations, as shown above. The order in which the ANSYS program evaluates an expression is as follows:

1. Operations in parentheses (innermost first)
2. Exponentiation (in order, from right to left)
3. Multiplication and division (in order, from left to right)
4. Unary association (such as +A or -A)
5. Addition and subtraction (in order, from left to right)
6. Logical evaluation (in order, from left to right)

Thus an expression such as Y2=A+B**C/D*E will be evaluated in this order: B**C first, /D second, *E third, and +A last. For clarity, you should use parentheses in expressions such as these. Parentheses can be nested up to four levels deep, and up to nine operations can be performed within each set of parentheses. As a general rule, avoid using blank spaces between operators in expressions. In particular, never include a blank space before the * character because the rest of the input line (beginning with the *) will be interpreted as a comment and therefore will be ignored. (Do not use this convention as a comment; use an exclamation point (!) for this purpose.)

### 3.8. Parametric Functions

A parametric function is a programmed sequence of mathematical operations which returns a single value, such as SIN(X), SQRT(B), and LOG(13.2). The following table provides a complete list of functions currently available in ANSYS.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS(x)</td>
<td>Absolute value of x.</td>
</tr>
<tr>
<td>SIGN(x,y)</td>
<td>Absolute value of x with sign of y. y=0 results in positive sign.</td>
</tr>
<tr>
<td>EXP(x)</td>
<td>Exponential of x (e^x).</td>
</tr>
<tr>
<td>LOG(x)</td>
<td>Natural log of x (ln(x)).</td>
</tr>
<tr>
<td>LOG10(x)</td>
<td>Common log of x (log_{10}(x)).</td>
</tr>
<tr>
<td>SQRT(x)</td>
<td>Square root of x.</td>
</tr>
<tr>
<td>NINT(x)</td>
<td>Nearest integer to x.</td>
</tr>
<tr>
<td>MOD(x,y)</td>
<td>Remainder of x/y, computed as x - (INT(x/y) * y). y=0 returns zero (0).</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>RAND(x,y)</td>
<td>Random number (uniform distribution) in the range x to y (x = lower bound, y = upper bound).</td>
</tr>
<tr>
<td>GDIS(x,y)</td>
<td>Random sample of a Gaussian (normal) distribution with mean x and standard deviation y.</td>
</tr>
<tr>
<td>SIN(x), COS(x), TAN(x)</td>
<td>Sine, Cosine, and Tangent of x. x is in radians by default, but can be changed to degrees with *AFUN.</td>
</tr>
<tr>
<td>SINH(x), COSH(x), TANH(x)</td>
<td>Hyperbolic sine, Hyperbolic cosine, and Hyperbolic tangent of x.</td>
</tr>
<tr>
<td>ASIN(x), ACOS(x), ATAN(x)</td>
<td>Arcsine, Arccosine, and Arctangent of x. x must be between -1.0 and +1.0 for ASIN and ACOS. Output is in radians by default, but can be changed to degrees with *AFUN. Range of output is -pi/2 to +pi/2 for ASIN and ATAN, and 0 to pi for ACOS.</td>
</tr>
<tr>
<td>ATAN2(y,x)</td>
<td>Arctangent of y/x with the sign of each component considered. Output is in radians by default, but can be changed to degrees with *AFUN. Range of output is -pi to +pi.</td>
</tr>
<tr>
<td>VALCHR (CPARM)</td>
<td>Numerical value of CPARM (if CPARM is non-numeric, returns 0.0).</td>
</tr>
<tr>
<td>CHRVAL (PARM)</td>
<td>Character value of numerical parameter PARM. Number of decimal places depends on magnitude.</td>
</tr>
<tr>
<td>UPCASE CPARM</td>
<td>Upper case equivalent of CPARM.</td>
</tr>
<tr>
<td>LWCASE (CPARM)</td>
<td>Lower case equivalent of CPARM.</td>
</tr>
</tbody>
</table>

The following are examples of parametric functions:

```
PI=ACOS(-1)             ! PI = arc cosine of -1, PI calculated to machine accuracy
Z3=COS(2*THETA)-Z1**2
R2=SQRT(ABS(R1-3))
X=RAND(-24,R2)          ! X = random number between -24 and R2
*AFUN,DEG               ! Units for angular functions are degrees
THETA=ATAN(SQRT(3))     ! THETA evaluates to 60 degrees
PHI=ATAN2(-SQRT(3),-1)  ! PHI evaluates to -120 degrees
*AFUN,RAD               ! Units for angular functions reset to radians
X249=NX(249)            ! X-coordinate of node 249
SLOPE=(KY(2)-KY(1))/(KX(2)-KX(1))   ! Slope of line joining keypoints 1 and 2
CHNUM=CHRVAL(X)         ! CHNUM = character value of X
UPPER=UPCASE(LABEL)     ! UPPER = uppercase character value of parameter LABEL
```

### 3.9. Saving, Resuming, and Writing Parameters

If you must use currently defined parameters in another ANSYS session, you can write them to a file and then read (resume) that file. When you read the file, you can either completely replace currently defined parameters or add to them (replacing those that already exist).

To write parameters to a file, use the PARSAV command (Utility Menu> Parameters> Save Parameters).

The parameters file is an ASCII file consisting largely of APDL *SET commands used to define the various parameters. The following example shows the format of this file.

```
/NOPR
*SET, A , 10.00000000000
*SET, B , 254.394875000
*SET, C , 'string '
*SET, _RETURN , 0.0000000000000E+00
*SET, _STATUS , 1.0000000000000
```
To read parameters from a file use the **PARRES** command (**Utility Menu**> **Parameters**> **Restore Parameters**).

If you wish, you can write up to ten parameters or array parameters using FORTRAN real formats to a file. You can use this feature to write your own output file for use in other programs, reports, etc. To do this, use the **VWRITE** command (**Utility Menu**> **Parameters**> **Array Parameters**> **Write to File**). The **VWRITE** command is discussed in *Operations Among Array Parameters* (p. 35).

### 3.10. Array Parameters

In addition to scalar (single valued) parameters, you can define array (multiple valued) parameters. ANSYS arrays can be:

- 1-D (a single column)
- 2-D (rows and columns)
- 3-D (rows, columns, and planes)
- 4-D (rows, columns, planes, and books)
- 5-D (rows, columns, planes, books, and shelves)

ANSYS provides three types of arrays:

**ARRAY**

This type is similar to FORTRAN arrays and is the default array type when dimensioning arrays. As with FORTRAN arrays, the indices for rows, columns, and planes are sequential integer numbers beginning with one. Array elements can be either integers or real numbers.

**CHAR**

This is a character array, with each element consisting of an alphanumeric value not exceeding eight characters. The indices for rows, columns, and planes are sequential integer numbers beginning with one.

**TABLE**

This is a special type of numeric array which allows ANSYS to calculate (through linear interpolation) values between these array elements explicitly defined in the array. Moreover, you can define the array indices for each row, column, and plane and these indices are real (not integer) numbers. Array elements can be either integers or real numbers. As we'll see in the later discussion on TABLE arrays, this capability provides a powerful method for describing mathematical functions.

**STRING**

You can use the **DIM**, STRING capability to enter character strings into your arrays. Index numbers for columns and planes are sequential values beginning with 1. Row indices are determined by the character position in the string. See the **DIM** command for more information.

All three types of arrays cannot exceed $2^{31}-1$ bytes. For a double precision array, each data item is 8 bytes, so the limit on number of entries is $(2^{31}-1)/8$.

The following array parameter topics are available:

- 3.10.1. Array Parameter Basics
- 3.10.2. Array Parameter Examples
- 3.10.3. TABLE Type Array Parameters
- 3.10.4. Defining and Listing Array Parameters
- 3.10.5. Specifying Array Element Values
3.10.1. Array Parameter Basics

Consider a 2-D array (either ARRAY or CHAR) as shown below. It is \( m \) rows long and \( n \) columns wide; that is, its dimensions are \( m \) times \( n \). Each row is identified by a row index number \( i \), which varies from 1 to \( m \), and each column is identified by a column index number \( j \), which varies from 1 to \( n \). The quantities that make up the array are array elements. Each array element is identified as \( (i, j) \), where \( i \) is its row index number and \( j \) is its column index number.

**Figure 3.1: A Graphical Representation of a 2-D Array**

We can extend these definitions to a 3-D array parameter, which may be \( m \) rows long, \( n \) columns wide, and \( p \) planes deep. The plane index number is \( k \), which varies from 1 to \( p \). Each array element is identified as \( (i, j, k) \). The following figure shows a 3-D array.

**Figure 3.2: A Graphical Representation of a 3-D Array**
3.10.2. Array Parameter Examples

Type ARRAY parameters consist of discrete numbers that are simply arranged in a tabular fashion for convenience. Consider the following examples.

\[ \begin{array}{c}
-47.6 \\
-5.2 \\
25.0 \\
86.5 \\
107.9 \\
188.7 \\
225.0 \\
\end{array} \quad \begin{array}{c}
0.025 \\
0.01 \\
0.265 \\
1.001 \\
0.832 \\
0.52 \\
0.002 \\
0.697 \\
0.01 \\
\end{array} \]
The parameter NTEMP could be an array of temperatures at selected nodes; NTEMP(1) = -47.6 could be the temperature at node 27, NTEMP(2) = -5.2 could be the temperature at node 43, and so on. Similarly, EVOLUM could be an array of element volumes, and COMPSTRS could be an array of nodal component stresses, with each column representing a particular direction (X, Y, Z, XY, YZ, XZ, for example).

A type CHAR array parameter is structured similarly to an ARRAY parameter, with the tabular values being alphanumeric character strings (up to eight characters). Two examples of character array parameters are:

```
[JOB1] [LOG]
[JOB2] [ERR]
[JOB3] [DB]
[JOB4] [LIB]
[JOB5] [MAC]
```

### 3.10.3. TABLE Type Array Parameters

A type TABLE array parameter consists of numbers (alphanumeric values are not valid) arranged in a tabular fashion, much like the ARRAY type. However, there are three important differences:

- ANSYS can calculate (through linear interpolation) any values that fall between the explicitly declared array element values.
- A table array contains a 0 row and 0 column used for data-access index values, and unlike standard arrays, these index values can be real numbers. The only restriction is that the index values must be numerically increasing (never decreasing) numbers. You must explicitly declare a data access index value for each row and column; otherwise the default value assigned is the "tiny number" (7.888609052E-31).
- You can more conveniently define the index starting point and index values via the *TAXIS command.
- A plane index value resides in the 0,0 location for each plane.

The following figure shows a TABLE array with data-access index values. Note that the indexes are specified as the "0" row and column values.
As shown in the above example, when configuring a table array you must set

- The plane index value as the 0,0 element value for each plane.
- The data-access column index values in the elements in the 0 row in plane 1. Only the column index values from plane 1 are used when accessing data from the array for all planes. When setting the array element values, you use the traditional row and column index numbers.
- The data-access row index values in the elements in the 0 column in plane 1. Only the row index values from plane 1 are used when accessing data from the array for all planes. When setting the array element values, you use the traditional row and column index numbers.

3.10.4. Defining and Listing Array Parameters

To define an array parameter, you must first declare its type and dimensions using the *DIM command (Utility Menu> Parameters> Array Parameters> Define/Edit).

This following examples illustrate the *DIM command used to dimension various types of arrays:

*DIM,AA,,4             ! Type ARRAY is default, dimension 4[x1x1]
*DIM,XYZ,ARRAY,12      ! Type ARRAY array, dimension 12[x1x1]
*DIM,FORCE,TABLE,5     ! Type TABLE array, dimension 5[x1x1]
*DIM,T2,,4,3           ! Dimensions are 4x3[x1]
*DIM,CPARR1,CHAR,5     ! Type CHAR array, dimension 5[x1x1]

Note

Array elements for ARRAY and TABLE are initialized to 0 (except for the 0 row and column for TABLE, which is initialized to the tiny value). Array elements for CHAR are initialized to a blank value.

For array parameter operations (for example *VOPER, *VFUN), the resulting array parameter (ParR) need not be dimensioned beforehand. Also, array parameters defined completely with the implied (colon) loops convention (for example a(1:5)=10,20,30,40,50) need not be dimensioned beforehand.
The next example shows how to fill a 5-D array with data. Use 1-D tables to load a 5-D table. Use the *TAXIS to define the table index values. See the full example at Example Analysis Using 5-D Table Array.

*dim,xval,array,X1
*dim,yval,array,Y1
yval(1)=0,20
*dim,zval,array,10
zval(1)=10,20,30,40,50,60,70,80,90,100
*dim,tval,array,5
tval(1)=1, .90, .80, .70, .60
*dim,tevl,array,5
tevl(1)=1,1.20,1.30,1.60,1.80
*dim,ccc,tab5,X1,Y1,Z1,D4,D5,X,Y,Z,TIME,TEMP
*taxis,ccc(1,1,1,1,1),1,0,wid                   !!! X-Dim
*taxis,ccc(1,1,1,1,1),2,0,hth                   !!! Y-Dim
*taxis,ccc(1,1,1,1,1),3,1,2,3,4,5,6,7,8,9,10    !!! Z-Dim
*taxis,ccc(1,1,1,1,1),4,0,10,20,30,40            !!! Time
*taxis,ccc(1,1,1,1,1),5,0,50,100,150,200        !!! Temp
*do,ii,1,2
  *do,jj,1,2
    *do,kk,1,10
      *do,ll,1,5
        *do,mm,1,5
          ccc(ii,jj,kk,ll,mm)=(xval(ii)+yval(jj)+zval(kk))*tval(ll)*tevl(mm)
          *endo
        *endo
      *endo
    *endo
  *endo
*endo

3.10.5. Specifying Array Element Values

You can specify array element values by

• Setting individual array element values through the *SET command or "=" shortcut.
• Filling individual vectors (columns) in the array with either specified or calculated values (the *VFILL command, for example).
• Interactively specifying values for the elements through the *VEDIT dialog box.
• Reading the values from an ASCII file (*VREAD or *TREAD commands).

You cannot create or edit 4- or 5-D arrays interactively. *VEDIT, *VREAD, and *TREAD are not applicable to 4- or 5-D arrays.

3.10.5.1. Specifying Individual Array Values

You can use either the *SET command or the "=" shortcut. Usage is the same as for scalar parameters, except that you now define a column of data (up to ten array element values per "=" command). For example, to define the parameter XYZ dimensioned above as a 12x1 array you will need two "=" commands. In the following example the first command defines the first eight array elements and the second command defines the next four array elements:

XYZ(1)=59.5,42.494,-9.01,-8.98,-8.98,9.01,-30.6,51
XYZ(9)=-51.9,14.88,10.8,-10.8
Notice that the starting location of the array element is indicated by the row index number of the parameter (1 in the first command, 9 in the second command).

The following example shows how to define the element values for the 4x3 array parameter T2, dimensioned earlier in the *DIM examples:

\[
\begin{bmatrix}
0.6 & 7.0 & 0.0002 \\
2.0 & 5.0 & -3.5 \\
-1.8 & 9.1 & 22.0 \\
4.0 & 62.5 & 0.01
\end{bmatrix}
\]

The following example defines element values for the TABLE array parameter FORCE discussed earlier.

\[
\begin{align*}
\text{FORCE} &= 0, 560, 560, 238.5, 0 \\
\text{FORCE} &= 1E-6, .8, 7.2, 8.5, 9.3
\end{align*}
\]

The first “=” command defines the five array elements of the TABLE array FORCE. The second and third “=” commands redefine the index numbers in the j=0 and i=0 row.

Character array parameters can also be defined using the "=" command. Assigned values can be up to eight characters each and must be enclosed in single quotes. For example:

\[
\begin{align*}
\text{*DIM,RESULT,CHAR,3} & \quad ! \text{Character array parameter with dimensions (3,1,1)} \\
\text{RESULT} &= 'SX','SY','SZ' & \quad ! \text{Assigns values to parameter RESULT}
\end{align*}
\]

Notice that, as when defining a numerical array parameter, the starting location of the array element must be specified (in this case, the row index number 1 is indicated).

\[\text{Note}\]

CHAR cannot be used as a character parameter name because it will create a conflict with the CHAR label on the *DIM command. ANSYS will substitute the character string value assigned to parameter CHAR when CHAR is input on the third field of the *DIM command (Type field).
3.10.5.2. Filling Array Vectors

You can use the *VFILL command (Utility Menu> Parameters> Array Parameters> Fill) to "fill" an ARRAY or TABLE vector (column).

See the *VFILL command reference information in the Command Reference for more detail about the command syntax. The following example illustrates the capabilities of the *VFILL command.

```
*DIM, DTAB, ARRAY, 4, 3                     ! dimension 4 x 3 numeric array
*VFILL, DTAB(1,1), DATA, -3, 8, -12, 57     ! four data values loaded into vector 1
*VFILL, DTAB(1,2), RAMP, 2.54, 2.54         ! fill vector 2 with values starting at
! 2.54 and incrementing by 2.54
*VFILL, DTAB(1,3), RAND, 1.5, 10            ! fill vector 3 with random numbers between
! 1.5 and 10. Results will vary due to
! random number generation.
```

DTAB =

```
-3  2.54  2.799901284
  8  5.08  6.11292418
-12 7.82  6.70205516
 57 10.16  4.11487684
```

3.10.5.3. Interactively Editing Arrays

The *VEDIT command (Utility Menu> Parameters> Array Parameters> Define/Edit), which is available only in interactive mode, launches a data entry dialog box you can use to edit an ARRAY or TABLE (not CHAR) array. The dialog box provides a number of convenient features:

- A spreadsheet-style editor for array element values.
- Navigational controls for scrolling through large arrays.
- An initialize function to set any row or column to a specified value (ARRAY type only).
- Delete, copy, and insert functions for moving rows or columns of data (ARRAY type only).

Complete instructions for using the dialog box are available from the box's Help button.

**Note**

You cannot edit a 4- or 5-D ARRAY or TABLE interactively.

Figure 3.5: An Example *VEDIT Dialog Box for an ARRAY
3.10.5.4. Filling an Array From a Data File Using *VREAD

You can fill an array from a data file using the *VREAD command (Utility Menu > Parameters > Array Parameters > Read from File). The command reads information from an ASCII data file and begins writing it into the array, starting with the index location that you specify. You can control the format of the information read from the file through data descriptors. The data descriptors must be enclosed in parenthesis and placed on the line following the *VREAD command. See Vector Operations (p. 35) for more information about data descriptors. The data descriptors control the number of fields to be read from each record, the width of the data fields, and the position of the decimal point in the field.

For example, given the following data file named dataval:

```
1.5     7.8   12.3
15.6  -45.6   42.5
```

and an array called EXAMPLE that has been dimensioned as 2 x 3, the following commands (provided as either a part or a macro or input listing)

```
*DIM,EXAMPLE,,2,3
*VREAD,EXAMPLE(1,1),dataval,,,JIK,3,2
(3F6.1)
```

result in

```
1.5     7.8   12.3
15.6  -45.6   42.5
```

The *VREAD command cannot be issued directly from the command input window. However, the Utility Menu > Parameters > Array Parameters > Read from File dialog box offers a way to specify the data descriptors and issue the command in interactive mode.

Note

You cannot fill a 4- or 5-D array using *VREAD.

3.10.5.5. Filling a TABLE Array From a Data File Using *TREAD

Once configured, you have two options for specifying values for the TABLE array elements: you can add values as you would for any other type of array, or you can read in a table of data from an external file.
To read in a table of data from an external file, you still define the TABLE array first, specifying the number of rows, columns, and planes, and the labels for each. You can then read an ASCII file containing the table of data using the *TREAD command (Utility Menu> Parameters> Array Parameters> Read from File). At this time, you also specify the number of lines to skip (NSKIP) between the top of the file and the first line of the table.

When reading data from an external file, remember:

- The file containing the table of data can be created in a text editor or an external application (such as Microsoft Excel), but it must be in ASCII form, tab-delimited, to be read into ANSYS.
- You must first define the array in ANSYS, remembering to allow for the index values (0,0).
- The values are read straight across the rows until all columns on each row of the array are filled; ANSYS then wraps from one row to the next and begins to fill those columns, and so on. Be sure that the dimensions of the array you defined are correct. If you mistakenly define fewer columns in the ANSYS array than required, ANSYS will start filling in the next row of the array using the values remaining in the first row of the data table being read. Similarly, if you define more columns in the ANSYS array than required, ANSYS will fill all columns of the array using values from the next row of the data table being read, and only then wrap and begin filling the next row.

You can create 1-D, 2-D, and 3-D tables by reading data from an external file. Examples of how you create each of these follows.

---

**Note**

You cannot fill a 4- or 5-D TABLE using *TREAD.

**Example 1: 1-D Table**

First, create the 1-D table using the application of your choice (such as a spreadsheet application, a text editor, etc.) and then save the file as a text file in tab-delimited format. In this example, the table is named "Tdata" and contains data for time vs. temperature. In its ASCII form, the table would look like this:

<table>
<thead>
<tr>
<th>Time</th>
<th>Temperature Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>70</td>
</tr>
<tr>
<td>4</td>
<td>75</td>
</tr>
</tbody>
</table>

In ANSYS, you define a TABLE parameter "Tt" using the *DIM command (Utility Menu> Parameters> Array Parameters> Define/Edit). Specify 4 rows and 1 column, row label of Time, and column label of Temp. Note that the data table you created has four rows and one column of data, plus the row and column index values (the first column - TIME - is the row index values) Then read in the file as described earlier, specifying 2 skipped lines. The TABLE array in ANSYS would look like this:
Figure 3.7: A Sample 1-D TABLE Array Dialog Box

This same example, done via command input, would look like the following:

*DIM, Tt, table, 4, 1, 1, TIME, TEMP
*TREAD, Tt, tdata, txt,, 2

Example 2: 2-D Table

For this example, create (in a spreadsheet application, a text editor, etc.) a 2-D table named "T2data" containing temperature data as a function of time and x-coordinate and read it into a TABLE array parameter called "Ttx." The table, in its ASCII form, would look like this:

<table>
<thead>
<tr>
<th>Temp (time-X-coord) Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

In ANSYS, you define a TABLE parameter "Ttx" using the *DIM command (Utility Menu> Parameters> Array Parameters> Define/Edit). Specify 4 rows, 5 columns, 1 plane, row label of TIME, and column label of X-COORD. Note that the data table you created has four rows and five columns of data, plus the row and column index values. Then read in the file as described earlier, specifying 2 skipped lines. The TABLE array in ANSYS would look like this:
Example 3: 3-D Table

For this example, create a 3-D table named "T3data" containing temperature data as a function of time, x-coordinate, and y-coordinate and read it into a TABLE array parameter called "Ttxy." The table, in its ASCII form, would look like this:

<table>
<thead>
<tr>
<th>Time</th>
<th>X-Coordinate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0  0 0 .3  .5  .7  .9</td>
</tr>
<tr>
<td>0</td>
<td>10 15 20 25 30</td>
</tr>
<tr>
<td>1</td>
<td>15 20 25 35 40</td>
</tr>
<tr>
<td>2</td>
<td>20 25 35 55 60</td>
</tr>
<tr>
<td>4</td>
<td>30 40 70 90 100</td>
</tr>
<tr>
<td>1.5</td>
<td>0  0 0 .3  .5  .7  .9</td>
</tr>
<tr>
<td>0</td>
<td>20 25 30 35 40</td>
</tr>
<tr>
<td>1</td>
<td>25 30 35 45 50</td>
</tr>
<tr>
<td>2</td>
<td>30 35 45 65 70</td>
</tr>
<tr>
<td>4</td>
<td>40 50 80 100 120</td>
</tr>
</tbody>
</table>

In the example above, the bold values (in the (0,0,Z) positions) indicate the separate planes. Each plane of data, along with the row and column index values, is repeated for the separate planes. Only the plane index value and the actual data values are different. The shaded area above shows the values that change from plane to plane.

In ANSYS, you define a TABLE parameter "Ttxy" using the *DIM command (Utility Menu> Parameters> Array Parameters> Define/Edit). In the case of a 3-D table, the table is dimensioned according to the number of rows, columns, and planes of data. The first column (TIME) is the row index values and the first row is the column index values. Specify 4 rows, 5 columns, 2 planes, row label of TIME, column label of X-COORD, and plane label of Y-COORD. Note that the data table you created has four rows and five columns.
of data in two planes, plus the row and column index values. Then read in the file as described earlier, specifying 2 skipped lines. The TABLE array in ANSYS would look like this for the second plane of data (Y=1.5):

**Figure 3.9: A Sample 3-D TABLE Array Dialog Box**

![Table Array Dialog Box](image)

This same example, done via command input, would look like the following:

*DIM,Ttxy,table,4,5,2,TIME,X-COORD,Y-COORD
*TREAD,Ttxy,t3data,txt,,2

### 3.10.5.6. Interpolating Values

When accessing information from the array, ANSYS will interpolate values between those explicitly set.

As examples of how ANSYS interpolates values in TABLE arrays, consider the following:

Given that \( A \) is a TABLE array parameter, the ANSYS program can calculate any value between \( A(1) \) and \( A(2) \), for example

- \( A(1.5) \) evaluates to 20.0 (halfway between 12.0 and 28.0)
- \( A(1.75) \) evaluates to 24.0
- \( A(1.9) \) evaluates to 26.4

Similarly, if \( PQ \) is a TABLE array parameter

- \( PQ(1.5,1) \) evaluates to -3.4 (halfway between 2.8 and -9.6)
- \( PQ(1,1.5) \) evaluates to 3.5 (halfway between 2.8 and 4.2)
- \( PQ(3.5,1.3) \) evaluates to 14.88

This feature allows you to describe a function, such as \( y=f(x) \), using a TABLE array parameter. You would use the \( j=0 \) column for values of the independent variable \( x \) and the "regular" \( j=1 \) column for values of \( y \). Consider, for example, a time-history forcing function described by five points as shown below.
Figure 3.10: Time-History Forcing Function

You can specify this function as a TABLE array parameter whose array elements are the force values, and whose row index numbers 1 through 5 are time values 0.0 through 9.3. Schematically, the parameter will then look like this:

\[
\begin{array}{c|c}
\text{Time} & \text{Force} \\
0.0 & 0.0 \\
0.8 & 560.0 \\
7.2 & 590.0 \\
8.5 & 238.5 \\
9.3 & 0.0 \\
\end{array}
\]

ANSYS can calculate (through linear interpolation) force values at times not specified in the FORCE parameter. For the above example, ANSYS will calculate a value of 89.4375 for FORCE(9). If a parameter location beyond the dimensions of the array is used, no extrapolation is done and the end value is used. For example, ANSYS will provide a value of 560.0 for FORCE(5,2) or 0.0 for FORCE(12).

You can see from these examples that TABLE array parameters can be very powerful tools in your analysis. Typical applications are time-history loading functions, response spectrum curves, stress-strain curves, material-versus-temperature curves, B-H curves for magnetic materials, and so forth. Be aware that TABLE array parameters require more computer time to process than the ARRAY type.

### 3.10.5.7. Retrieving Values into or Restoring Array Parameter Values

You can use the `*VGET` command (Utility Menu> Parameters> Get Array Data), which is similar to `*GET`, to retrieve ANSYS supplied values and store them in an array.

You must define a starting array location number for the array parameter the `*VGET` command creates. Looping continues over successive entity numbers for the `KLOOP` default. For example, `*VGET,A(1),ELEM,5,CENT,X` returns the centroid x-location of element 5 and stores the result in the first location of A. Retrieving continues with elements 6, 7, and so on until successive array locations are filled. In this example, if `KLOOP` is 4, then the centroid of x, y, and z are returned.

To restore array parameter values, use the `*VPUT` command (Utility Menu> Parameters> Array Operations> Put Array Data).

The `*VPUT` command uses the same arguments as the `*VGET` command (described above), but does the opposite of the `*VGET` operation. For a list of valid labels for `*VPUT` items, see the command's description in the Command Reference.

The ANSYS program "puts" vector items directly, without any coordinate system transformation. `*VPUT` can replace existing array items, but can't create new items. Degree of freedom results that are changed in the
database are available for all subsequent operations. Other results change temporarily, and are available mainly for immediately following print and display operations.

### Note

Use this command with extreme caution, as it can alter entire sections of the database. The **VPUT** command doesn’t support all items on the **VGET** item list because putting values into some locations could make the ANSYS database inconsistent.

### 3.10.5.8. Listing Array Parameters

As with scalar parameters, you can use the **STATUS** command to list array parameters. The following examples illustrate the **STATUS** command in use:

```
*STATUS
ABBREVIATION STATUS-

ABBREVI  STRING
  SAVE_DB  SAVE
  RESUM_DB  RESUME
  QUIT     Fnc_/EXIT
  POWRGRPH Fnc_/GRAPHICS
  ANSYSWEB Fnc_/HomePage

PARAMETER STATUS- ( 5 PARAMETERS DEFINED)
  (INCLUDING 2 INTERNAL PARAMETERS)

  NAME         VALUE         TYPE         DIMENSIONS
  MYCHAR      hi        CHARACTER
  MYPAR                     ARRAY       4       6       1
  MYPAR1    .987350000      SCALAR

*STATUS,XYZ(1),5,9 ! Lists rows 5 through 9 of xyz
PARAMETER STATUS- XYZ ( 4 PARAMETERS DEFINED)

  LOCATION          VALUE
  5     1     1   -8.98000000
  6     1     1    9.01000000
  7     1     1   -30.6000000
  8     1     1    51.0000000
  9     1     1   -51.9000000

*STATUS,FORCE(1),,,0 ! Lists parameter force, includes j=0 column
PARAMETER STATUS- FORCE ( 4 PARAMETERS DEFINED)

  LOCATION          VALUE
  1     0     1   0.000000000E+00
  2     0     1   0.800000000
  3     0     1    7.20000000
  4     0     1    8.50000000
  5     0     1    9.30000000
  1     1     1  0.000000000E+00
  2     1     1    560.000000
  3     1     1    560.000000
  4     1     1    238.500000
  5     1     1  0.000000000E+00

*STATUS,T2(1,1) ! Lists parameter T2
PARAMETER STATUS- T2 ( 4 PARAMETERS DEFINED)

  LOCATION          VALUE
  1     1     1   0.600000000
  2     1     1    2.00000000
  3     1     1   -1.80000000
  4     1     1    4.00000000
  1     2     1    7.00000000
```
2     2     1    5.00000000
3     2     1    9.10000000
4     2     1    62.50000000
1     3     1    2.00000000E-04
2     3     1   -3.50000000
3     3     1    22.00000000
4     3     1   1.00000000E-02

*STATUS,RESULT(1)!Lists parameter RESULT

PARAMETER STATUS- RESULT     ( 4 PARAMETERS DEFINED)
LOCATION     VALUE
1     1     1     SX(CHAR)
2     1     1     SY(CHAR)
3     1     1     SZ(CHAR)

3.10.6. Writing Data Files

You can write formatted data files (tabular formatting) from data held in arrays through the *VWRITE command. The command takes up to 10 array vectors as arguments and writes the data contained in those vectors to the currently open file (*CFOPEN command). The format for each vector is specified with FORTRAN data descriptors on the line following the *VWRITE command (therefore you can't issue the *VWRITE command from the ANSYS input window.)

An array vector, specified with a starting element location (such as MYARRAY(1,2,1)). You can also use an expression, which is evaluated as a constant value for that field in each row of the data file. The keyword SEQU evaluates to a sequential column of integers, starting from one.

The format of each row in the data file is determined by the data descriptor line. You must include one descriptor for each argument to the command. Do not include the word FORMAT in the descriptor line. You can use any real format or character format descriptor; however, you may not use either integer or list directed descriptors.

3.10.6.1. Format Data Descriptors

If you aren't familiar with FORTRAN data descriptors, this section will get you started with formatting your data file. For more information, consult the documentation for the FORTRAN compiler for your particular platform.

You must provide a data descriptor for each data item you specify as an argument to the *VWRITE command. In general, you can use the F descriptor (floating point) for any numeric values. The F descriptor takes the syntax

\[ Fw.d \]

where

\[ w \]

Is the width of the data field in characters.

\[ d \]

Is the number of digits to the right of the decimal point.

Thus, for a field that is 10 characters wide and has eight characters after the decimal point, you would use the following data descriptor:

\[ F10.8 \]
For character fields, you can use the A descriptor. The A descriptor has the syntax

\[ A_w \]

where

\[ w \]

Is the width of the data field in characters.

Thus, for a character field that is eight characters wide, the descriptor is \( A_8 \)

The following examples illustrate the \(*VWRITE\) command and data descriptors in use.

Given that the MYDATA array has been dimensioned and filled with the following values:

\[
\begin{array}{ccccccc}
2.15215183 & 3.80075020 & 5.28636971 & 7.15706483 & 13.7859423 & 87.4970443 & \\
2.30485343 & 4.44486730 & 5.40919563 & 7.68192625 & 15.5483202 & 86.5677915 & \\
2.01051819 & 3.39165246 & 5.93668087 & 7.38584253 & 18.4639865 & 45.7259886 & \\
2.36833012 & 3.32711472 & 5.63220304 & 7.22482004 & 18.7977889 & 39.7902425 & \\
2.22706343 & 3.48214546 & 5.54685145 & 7.90325139 & 14.0708891 & 37.6008897 & \\
\end{array}
\]

The following short macro first defines the scalar parameter X as having a value of 25 and then opens the file vector (**CFOPEN command). The **VWRITE command then defines the data to be written to the file. In this case, the first vector written uses the SEQU keyword to provide row numbers. Note that in some cases that constants, scalar parameters, and operations that include array element values are written to the file. Note the data file contents for these items.

\[ x=25 \]

\[*cfopen, vector\]

\[*vwrite, SEQU, mydata(1,1,1), mydata(1,2,1), mydata(1,3,1), 10.2, x, mydata(1,1,1)+3\]


\[*cfclose\]

The macro creates the following data file:

1. \[1.2.1522 \quad 3.9 \quad 5.286370 \quad 10.2 \quad 25. \quad 5.2\]
2. \[2.2.3049 \quad 4.0 \quad 5.409196 \quad 10.2 \quad 25. \quad 5.2\]
3. \[3.2.0105 \quad 3.4 \quad 5.936638 \quad 10.2 \quad 25. \quad 5.2\]
4. \[4.2.3683 \quad 3.3 \quad 5.632203 \quad 10.2 \quad 25. \quad 5.2\]
5. \[5.2.8491 \quad 4.8 \quad 5.976023 \quad 10.2 \quad 25. \quad 5.2\]
6. \[6.2.2280 \quad 3.5 \quad 5.546851 \quad 10.2 \quad 25. \quad 5.2\]

The second example uses the following previously dimensioned and filled array:

\[
\begin{array}{ccc}
10 & 50 \\
20 & 70 \\
30 & 80
\end{array}
\]

Note the use of descriptors in the following example **VWRITE command:

\[*vwrite, SEQU, mydata(1,1), mydata(2,1), (mydata(1,1)+mydata(1,2))\]

\[(' Row', F3.0, ' contains ', 2F7.3, ' Is their sum ', F7.3, '?')\]

The resulting data file is

\[
\begin{align*}
\text{Row 1. contains} & \quad 10.000 \quad 50.000. \quad \text{Is their sum} \quad 60.000 \quad ? \\
\text{Row 2. contains} & \quad 20.000 \quad 60.000. \quad \text{Is their sum} \quad 60.000 \quad ? \\
\text{Row 3. contains} & \quad 30.000 \quad 70.000. \quad \text{Is their sum} \quad 60.000 \quad ?
\end{align*}
\]
3.10.7. Operations Among Array Parameters

Just as parametric expressions and functions allow operations among scalar parameters, a series of commands is available to perform operations among array parameters. There are classes of operations: operations on columns (vectors), known as vector operations and operations on entire matrices (arrays), known as matrix operations. All operations are affected by a set of specification commands, which are discussed in Specification Commands for Vector and Matrix Operations (p. 39).

3.10.7.1. Vector Operations

Vector operations are simply a set of operations - addition, subtraction, sine, cosine, dot product, cross product, etc. - repeated over a sequence of array elements. Do-loops (discussed in Looping: Do-Loops) can be employed for this purpose, but a more convenient and much faster way is to use the vector operation commands - *VOPER, *VFUN, *VSCFUN, *VITRP, *VFILL, *VREAD, and *VGET. Of these listed vector operation commands, only *VREAD and *VWRITE are valid for character array parameters. Other vector operation commands apply only to array parameters dimensioned (*DIM) as ARRAY type or TABLE type.

The *VFILL, *VREAD, *VGET, *VWRITE, and *DIM commands were introduced earlier in this chapter. Other commands that are discussed in this section include

*VOPER or Utility Menu> Parameters> Array Operations> Vector Operations
Performs an operation on two input array vectors and produces a single output array vector.

*VFUN or Utility Menu> Parameters> Array Operations> Vector Functions
Performs a function on a single input array vector and produces a single output array vector.

*VSCFUN or Utility Menu> Parameters> Array Operations> Vector-Scalar Func
Determines the properties of a single input array vector and places the result in a specified scalar parameter.

*VITRP or Utility Menu> Parameters> Array Operations> VectorInterpolate
Forms an array parameter (type ARRAY) by interpolating an array parameter (type TABLE) at specified table index locations.

The examples below illustrate the use of some of these commands. Refer to the Command Reference for syntactical information about these commands. For all of the following examples, the array parameters (of type ARRAY) X, Y, and THETA have been dimensioned and defined.

X =
| -2 6 8 0 |
| 1 0 2 12 |
| 4 -3 -1 7 |
| -8 1 10 -5 |

Y =
| 3 2 5 -6 |
| -5 -7 1 0 |
| 8 0 0 11 |
| 1 4 9 16 |

THETA =
| 0 |
| 15 |
| 30 |
| 45 |
| 60 |
| 75 |
| 90 |

In the following example, the result array is first dimensioned (Z1). The *VOPER command then adds column 2 of X to column 1 of Y, both starting at row 1, and then places the result into Z1. Notice that the starting location (the row and column index numbers) must be specified for all array parameters. The operation then progresses sequentially down the specified vector.

*DIM, Z1, ARRAY, 4
*VOPER, Z1(1), X(1,2), ADD, Y(1,1)
In the following example, again the result array (Z2) is dimensioned first. The *VOPER command then multiplies the first column of X (starting at row 2) with the fourth column of Y (starting at row 1) and writes the results to Z2 (starting at row 1).

\[
\begin{align*}
*\text{DIM}, & \, \text{Z2}, \, \text{ARRAY}, \, 3 \\
*\text{VOPER}, & \, \text{Z2(1)}, \, \text{X(2,1)}, \, \text{MULT}, \, \text{Y(1,4)}
\end{align*}
\]

\[
\begin{bmatrix}
-6 \\
0 \\
-88
\end{bmatrix}
\]

In this example, again the results array (Z4) is dimensioned first. The *VOPER command then performs the cross product of four pairs of vectors, one pair for each row of X and Y. The \(i\), \(j\), and \(k\) components of these vectors are columns 1, 2, and 3 respectively of X and columns 2, 3, and 4 of Y. The results are written to Z4, whose \(i\), \(j\), and \(k\) components are vectors 1, 2, and 3 respectively.

\[
\begin{align*}
*\text{DIM}, & \, \text{Z4}, \, \text{ARRAY}, \, 4, \, 3 \\
*\text{VOPER}, & \, \text{Z4(1,1)}, \, \text{X(1,1)}, \, \text{CROSS}, \, \text{Y(1,2)}
\end{align*}
\]

\[
\begin{bmatrix}
-76 & 4 & -22 \\
-2 & -14 & 1 \\
-33 & -44 & 0 \\
-74 & 168 & -76
\end{bmatrix}
\]

In the following example, the results array (A3) is dimensioned first. The *VFUN command then raises each element in vector 2 of X to the power of 2 and writes the results to A3.

\[
\begin{align*}
*\text{DIM}, & \, \text{A3}, \, \text{ARRAY}, \, 4 \\
*\text{VFUN}, & \, \text{A3(1)}, \, \text{PWR}, \, \text{X(1,2)}, \, 2
\end{align*}
\]

\[
\begin{bmatrix}
36 \\
0 \\
9 \\
1
\end{bmatrix}
\]

In this example, the results array (A4) is dimensioned. The two *VFUN commands then calculate the cosine and sine of array elements in THETA and place the results in the first and second columns, respectively, of A4. Notice that A4 now represents a circular arc spanning 90°, described by seven points (whose \(x\), \(y\), and \(z\) global Cartesian coordinates are the three vectors). The arc has a radius of 1.0 and lies parallel to the x-y plane at \(z = 2.0\).

\[
\begin{align*}
*\text{DIM}, & \, \text{A4}, \, \text{ARRAY}, \, 7, \, 3 \\
*\text{AFUN}, & \, \text{DEG} \\
*\text{VFUN}, & \, \text{A4(1,1)}, \, \text{COS}, \, \text{THETA(1)} \\
*\text{VFUN}, & \, \text{A4(1,2)}, \, \text{SIN}, \, \text{THETA(1)}
\end{align*}
\]

\[
\begin{bmatrix}
1.0 & 0.0 & 2.0 \\
0.966 & 0.259 & 2.0 \\
0.866 & 0.5 & 2.0 \\
0.707 & 0.707 & 2.0 \\
0.5 & 0.866 & 2.0 \\
0.259 & 0.966 & 2.0 \\
0.0 & 1.0 & 2.0
\end{bmatrix}
\]
In this example, the results array (A5) is first dimensioned. Then, the *VFUN command calculates the tangent vector at each point on the curve represented by A4, normalizes it to 1.0, and places the results in A5.

```
*DIM, A5, ARRAY, 7, 3
*VFUN, A5(1,1), TANG, A4(1,1)
```

\[
\begin{bmatrix}
-0.131 & 0.991 & 0 \\
-0.259 & 0.985 & 0 \\
-0.5 & 0.866 & 0 \\
-0.707 & 0.707 & 0 \\
-0.866 & 0.5 & 0 \\
-0.966 & 0.259 & 0 \\
-0.991 & 0.131 & 0
\end{bmatrix}
\]

A5

Two additional *VOPER operations, gather (GATH) and scatter (SCAT), are used to copy values from one vector to another based on numbers contained in a "position" vector. The following example demonstrates the gather operation. Note that, as always, the results array must be dimensioned first. In the example, the gather operation copies the value of B1 to B3 (using the index positions specified in B2). Note that the last element in B3 is 0 as this is its initialized value.

```
*DIM, B1,, 4
*DIM, B2,, 3
*DIM, B3,, 4
B1(1)=10, 20, 30, 40
B2(1)=2, 4, 1
*VOPER, B3(1), B1(1), GATH, B2(1)
```

\[
\begin{bmatrix}
20 \\
40 \\
10 \\
0
\end{bmatrix}
\]

B3

### 3.10.7.2. Matrix Operations

Matrix operations are mathematical operations between numerical array parameter matrices, such as matrix multiplication, calculating the transpose, and solving simultaneous equations.

Commands discussed in this section include

**MOPER** or **Utility Menu> Parameters> Array Operations> Matrix Operations**

Performs matrix operations on two input array parameter matrices and produces one output array parameter matrix. Matrix operations include:

- Matrix multiplication
- Solution of simultaneous equations
- Sorting (in ascending order) on a specified vector in a matrix
- Covariance between two vectors
- Correlation between two vectors

**MFUN** or **Utility Menu> Parameters> Array Operations> Matrix Functions**

Copies or transposes an array parameter matrix (accepts one input matrix and produces one output matrix).

**MFOURI** or **Utility Menu> Parameters> Array Operations> Matrix Fourier**

Calculates the coefficients for or evaluates a Fourier series.

The examples below illustrate the use of some of these commands. Refer to the *Command Reference* for syntactical information about these commands.
This example shows the sorting capabilities of the *MOPER command. For this example, assume that the array (SORTDATA) has been dimensioned and its element values have been defined as follows:

\[
\begin{bmatrix}
3 & 10 & 11 \\
5 & 4 & 12 \\
8 & -9 & 13 \\
2 & 7 & 14 \\
6 & 1 & 15 \\
\end{bmatrix}
\]

First, the OLDORDER array is dimensioned. The *MOPER command will place the original order of the rows into OLDORDER. The *MOPER command then sorts the rows in SORTDATA so that the 1,1 vector is now in ascending order.

*dim,oldorder,,5
*moper,oldorder(1),sortdata(1,1),sort,sortdata(1,1)

The following array values result from the *MOPER command:

\[
\begin{bmatrix}
2 & 7 & 14 \\
3 & 10 & 11 \\
5 & 4 & 12 \\
8 & 1 & 15 \\
8 & -9 & 13 \\
\end{bmatrix}
\]

To put the SORTDATA array back into its original order, you could then issue the following command:

*moper,oldorder(1),sortdata(1,1),sort,oldorder(1,1)

In the following example, the *MOPER command solves a set of simultaneous equations. The following two arrays have been dimensioned and their values assigned:

\[
A = \begin{bmatrix}
2 & 4 & 3 & 2 \\
3 & 6 & 5 & 2 \\
2 & 5 & 2 & -3 \\
4 & 5 & 14 & 14 \\
\end{bmatrix}
\quad B = \begin{bmatrix}
2 \\
2 \\
3 \\
11 \\
\end{bmatrix}
\]

The *MOPER command can solve a set of simultaneous equations for a square matrix. The equations take the form

\[a_{n1}X_1 + a_{n2}X_2 + \ldots + a_{nn}X_n = b_n\]

In the case of the above arrays, the *MOPER command will solve the following set of simultaneous equations:

\[
2X_1 + 4X_2 + 3X_3 + 2X_4 = 2 \\
3X_1 + 6X_2 + 5X_3 + 2X_4 = 2 \\
2X_1 + 5X_2 + 2X_3 - 3X_4 = 3 \\
4X_1 + 5X_2 + 14X_3 + 14X_4 = 11
\]

To solve the equations, first the results array (C) is dimensioned. Then the *MOPER command solves the equations, using A as the matrix of a coefficients and B as a vector of b values.

*DIM,C,,4
*MOPER,C(1),A(1,1),SOLV,B(1)

The C array now contains the following solutions.
The following example shows the *MFUN command used to transpose data in an array. For this example, assume that the array (DATA) was dimensioned and filled with the following values:

\[
C = \begin{bmatrix}
-68 \\
26 \\
6 \\
4
\end{bmatrix}
\]

As always, the results array (DATATRAN) is dimensioned first, then the *MFUN command transposes the values and writes them to DATATRAN.

\[
\text{DATA} = \begin{bmatrix}
34 & 25 \\
22 & 68 \\
-7 & 12
\end{bmatrix}
\]

The following shows the results in the DATATRAN array:

\[
\text{DATATRAN} = \begin{bmatrix}
34 & 22 & -7 \\
25 & 68 & 12
\end{bmatrix}
\]

### 3.10.7.3. Specification Commands for Vector and Matrix Operations

All the vector and matrix operation commands are affected by the setting of the following specification commands: *VCUM, VABS, VFACT, VLEN, VCOL, and VMASK. (Of all specification commands, only VLEN and VMASK, in conjunction with VREAD or VWRITE, are valid for character array parameters.) You can check the status of these commands with the *VSTAT command. Most of these commands (and their corresponding GUI paths) were introduced earlier in this chapter. The others are explained in the following.

With the exception of the *VSTAT command, which you cannot access directly in the GUI, all of the specification commands described below are available via menu path Utility Menu> Parameters> Array Operations> Operation Settings.

**Important:** All specification commands are reset to their default settings after each vector or matrix operation.

The following lists the available array specification commands:

**VCUM**

Specifies whether results will be cumulative or noncumulative (overwriting previous results). ParR, the result of a vector operation, is either added to an existing parameter of the same name or overwritten. The default is noncumulative results, that is, ParR overwrites an existing parameter of the same name.

**VABS**

Applies an absolute value to any or all of the parameters involved in a vector operation. The default is to use the real (algebraic) value.

**VFACT**

Applies a scale factor to any or all of the parameters involved in a vector operation. The default scale factor is 1.0 (full value).

**VCOL**

Specifies the number of columns in matrix operations. The default is to fill all locations of the result array from the specified starting location.
*VSTAT
Lists the current specifications for the array parameters.

*VLEN or Utility Menu> Parameters> Array Operations> Operation Settings
Specifies the number of rows to be used in array parameter operations.

*VMASK or Utility Menu> Parameters> Array Operations> Operation Settings
Specifies an array parameter as a masking vector.

The following table lists the various specification commands and the vector and matrix array commands that they affect.

<table>
<thead>
<tr>
<th>Command</th>
<th>*VABS</th>
<th>*VFACT</th>
<th>*VCUM</th>
<th>*VCOL</th>
<th>*VLEN,NROW,NINC</th>
<th>*VMASK</th>
</tr>
</thead>
<tbody>
<tr>
<td>*MFOURI</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>*MFUN</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>*MOPER</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>*VFILL</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>*VFUN</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>*VGET</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>*VITRP</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>*VOPER</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>VPLOT</td>
<td>No</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>*VPUT</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>*VREAD</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>*VSCFUN</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>*VWRITE</td>
<td>No</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The examples below illustrate the use of some of the specification commands. Refer to the Command Reference for syntactical information about these commands.

In the following, the results array (CMPR) is dimensioned. The two *VFUN commands, in conjunction with the preceding *VMASK and *VLEN commands, then compress selected data and write them to specified locations in CMPR. The complement to the COMP operation is the EXPA operation on the *VFUN command.

```
*DIM,CMPR,ARRAY,4,4
*VLEN,4,2! Do next *V---- operation on four rows,
! skipping every second row
*VFUN,CMPR(1,2),COMP,Y(1,1)
*VMASK,X(1,3)! Use column 3 of X as a mask for next *V----
! operation
*VFUN,CMPR(1,3),COMP,Y(1,2)
```

```
CMPR = 
[ 0 3 2 0 ]
[ 0 8 -7 0 ]
[ 0 0 4 0 ]
[ 0 0 0 0 ]
```

This example uses the *VFACT command to round the values in an array vector to the number of decimal places specified by the NUMDP scalar parameter (set to 2 in the example). The NUMDATA array has been dimensioned and filled with the following values:
This example uses the *VLEN and *VMASK commands to find the set of prime numbers less than 100. An array, MASKVECT, is created using 1.0 to indicate that the row value is a prime number and 0.0 to indicate that the value isn’t prime. The algorithm used to create the mask vector is to initialize all rows whose value is greater than 1 to 1.0 and then loop through the range of possible factors, eliminating all multiples of the factor. The *VLEN command sets the row increment for performing operations to FACTOR. When the *VFILL command is processed, the row number is incremented by this value. Because the starting row is FACTOR x 2, the rows are processed by each loop in the following manner: FACTOR x 2, FACTOR x 3, FACTOR x 4, etc.

The resultant output from the *STATUS command, showing the first 10 elements in NUMBERS is:

**PARAMETER STATUS-- NUMBERS ( 5 PARAMETERS DEFINED)**

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>
3.10.8. Plotting Array Parameter Vectors

You can graphically display array vector values using the *VPLOT command.

The following demonstrates some of the capabilities of the *VPLOT command. For this example, two TABLE arrays (TABLEVAL and TABLE) and one numeric array have been dimensioned and filled with the following values:

\[
\text{TABLEVAL} = \begin{bmatrix} 4 & 6 & 12 \\ 7 & 8 & 6 \\ 15 & 10 & 3 \end{bmatrix}, \quad \text{TABLE} = \begin{bmatrix} 19 & 70 \\ 88 & 80 \\ 99 & 95 \end{bmatrix}
\]

\[
\text{ARRAYVAL} = \begin{bmatrix} 6 & 12 \\ 8 & 6 \\ 10 & 3 \end{bmatrix}
\]

The following are example *VPLOT commands and their resulting plots. Note that since ARRAY data is unordered it is plotted as a histogram; TABLE data is ordered and is therefore plotted as a curve.

**Figure 3.11: Sample Plot Using *VPLOT, \(,\text{arrayval}(1,1),2\)**

**Figure 3.12: Sample Plot Using *VPLOT, \(,\text{tableval}(1,1),2\)**
3.10.9. Modifying Curve Labels

When you use *VPLOT to create your curves, default labels are assigned. Normally, the label for curve 1 is “COL 1”, the label for curve 2 is “COL 2” and so on; the column number is the field containing the dependent variables for that particular curve. You can use the /GCOLUMN command to apply your own labels to the curves (any string of up to eight characters).

The example below uses the /GCOLUMN command at the beginning of the program input to apply the labels “string01” and “string02” to the array curve.

```
/gcol,1,string01
/gcol,2,string02

*dim,xxx,array,10
*dim,yyy,array,10,2

xxx(1,1) = 1e6
xxx(2,1) = 1e6 + 1e5
xxx(3,1) = 1e6 + 2e5
xxx(4,1) = 1e6 + 3e5
xxx(5,1) = 1e6 + 4e5
xxx(6,1) = 1e6 + 5e5
xxx(7,1) = 1e6 + 6e5
xxx(8,1) = 1e6 + 7e5
xxx(9,1) = 1e6 + 8e5
```
\texttt{xxx(10,1) = 1e6 + 9e5}

\texttt{yyy(1,1) = 1}
\texttt{yyy(2,1) = 4}
\texttt{yyy(3,1) = 9}
\texttt{yyy(4,1) = 16}
\texttt{yyy(5,1) = 25}
\texttt{yyy(6,1) = 36}
\texttt{yyy(7,1) = 49}
\texttt{yyy(8,1) = 64}
\texttt{yyy(9,1) = 81}
\texttt{yyy(10,1) = 100}

\texttt{yyy(1,2) = 1}
\texttt{yyy(2,2) = 2}
\texttt{yyy(3,2) = 3}
\texttt{yyy(4,2) = 4}
\texttt{yyy(5,2) = 5}
\texttt{yyy(6,2) = 6}
\texttt{yyy(7,2) = 7}
\texttt{yyy(8,2) = 8}
\texttt{yyy(9,2) = 9}
\texttt{yyy(10,2) = 10}

\texttt{*vplo,xxx(1,1), yyy(1,1) ,2}

\textbf{Figure 3.15: Sample Plot With User-specified Labels}

The labels can be returned to the default value (COL 1 and COL 2) by issuing the \texttt{/GCOLUMN} command with no string specified.

\texttt{/gcol,1}
\texttt{/gcol,2}
Chapter 4: APDL as a Macro Language

You can record a frequently used sequence of ANSYS commands in a macro file (these are sometimes called command files). Creating a macro enables you to, in effect, create your own custom ANSYS command. For example, calculating power loss due to eddy currents in a magnetic analysis would require a series of ANSYS commands in the postprocessor. By recording this set of commands in a macro, you have a new, single command that executes all of the commands required for that calculation. In addition to executing a series of ANSYS commands, a macro can call GUI functions or pass values into arguments.

You can also nest macros. That is, one macro can call a second macro, the second macro can call a third macro, and so on. You can use up to 20 nesting levels, including any file switches caused by the ANSYS /INPUT command. After each nested macro executes, the ANSYS program returns control to the previous macro level.

The following is a very simple example macro file. In this example, the macro creates a block with dimensions 4, 3, and 2 and a sphere with a radius of 1. It then subtracts the sphere from one corner of the block.

```
/prep7
/view,-1,-2,-3
/block,4,3,2
/sphere,1
/vsbv,1,2
/finish
```

If this macro were called mymacro.mac, you could execute this sequence of commands with the following single ANSYS command

```
*use,mymacro
```

or (because the extension is .mac)

```
mymacro
```

Although this is not a realistic macro, it does illustrate the principle.

This chapter provides information on the various ways you can create, store, and execute macros. It also discusses the basic information you need to use APDL as a scripting language in creating macros. APDL commands used to define and execute macros are listed in Chapter 2 of the Command Reference.

The following specific macro topics are available:

4.1. Creating a Macro  
4.2. Executing Macros and Macro Libraries  
4.3. Local Variables  
4.4. Controlling Program Flow in APDL  
4.5. Control Functions Quick Reference  
4.6. Using the _STATUS and _RETURN Parameters in Macros  
4.7. Using Macros with Components and Assemblies  
4.8. Reviewing Example Macros
4.1. Creating a Macro

You can create macros either within ANSYS itself or using your text editor of choice (such as emacs, vi, or wordpad). If your macro is fairly simple and short, creating it in ANSYS can be very convenient. If you are creating a longer, more complex macro or editing an existing macro then you will need a text editor. Also, using a text editor allows you to use a similar macro or ANSYS log file as the source for your macro.

For any long, complex macro you should always consider either using a similar macro as a starting point or running the task interactively in ANSYS and using the resulting log file as the basis of your macro. Either method can greatly reduce the time and effort required to create a suitable macro.

The following creating macro topics are available:
- 4.1.1. Macro File Naming Conventions
- 4.1.2. Macro Search Path
- 4.1.3. Creating a Macro Within ANSYS
- 4.1.4. Creating Macros with a Text Editor
- 4.1.5. Using Macro Library Files

4.1.1. Macro File Naming Conventions

Macros are a sequence of ANSYS commands stored in a file. Macros should not have the same name as an existing ANSYS command, or start with the first four characters of an ANSYS command, because ANSYS will execute the internal command instead of the macro. The following naming restrictions apply to macro files:

- The file name cannot exceed 32 characters.
- The file name cannot begin with a numeral.
- The file extension cannot contain more than eight characters (if you are executing the macro as if it were an ANSYS command it should have the extension .mac.)
- The file name or extension cannot contain spaces.
- The file name or extension cannot contain any characters prohibited by your file system and for portability should not contain any characters prohibited by either UNIX or Windows file systems.

To ensure that you are not using the name of an ANSYS command, before creating a macro try running the file name that you wish to use as an ANSYS command. If ANSYS returns the message shown below, you will know that the command is not used in the current processor. You should check the macro file name in each processor in which you plan to use the macro. (You could also check if the macro file name matches any command listed in the online documentation; however, this method cannot locate the names of undocumented commands.)

![Figure 4.1: ANSYS Message Box for Unknown Command](image)

Using the .mac extension allows ANSYS to execute the macro as it would any internal command. You should avoid using the extension .MAC because it is used for ANSYS internal macros.
4.1.2. Macro Search Path

By default, ANSYS searches for a user macro file (.mac extension) in the following locations:

1. The /ansys_inc/v120/ansys/apdl directory.
2. The directory (or directories) designated by the ANSYS_MACROLIB environment variable (if defined) or the login (home) directory. This environment variable is documented in The ANSYS Environment chapter of the Operations Guide.
3. The directory designated by the $HOME environment variable.
4. The working directory.

You can place macros for your personal use in your home directory. Macros that should be available across your site should be placed in the /ansys_inc/v120/ansys/apdl directory or some commonly accessible directory that everyone can reference through the ANSYS_MACROLIB environment variable.

For Windows users: The "current directory" is the default directory (usually a network resource) set by administrators and you should ask your network administrator for its location. You can use environment variables to create a local "home directory." The local home directory is checked after the default directory designated in your domain profile.

4.1.3. Creating a Macro Within ANSYS

You can create a macro by four methods from within ANSYS:

- Issue the *CREATE command in the input window. Parameter values are not resolved and parameter names are written to the file.
- Use the *CFOPEN, *CFWRITE, and *CFCLOS commands. Parameter names are resolved to their current values and those values are written to the macro file.
- Issue the /TEE command in the input window. This command writes a list of commands to a file at the same time that the commands are being executed. As the commands are executed in the current ANSYS session, parameter names are resolved to their current values. However, in the file that is created, parameter values are not resolved and parameter names are written instead.
- Choose the Utility Menu> Macro> Create Macro menu item. This method opens a dialog box that can be used as a simple, multiline editor for creating macros. Parameter values are not resolved and parameter names are written to the file.

The following sections detail each of these methods.

4.1.3.1. Using *CREATE

Issuing *CREATE redirects ANSYS commands entered in the command input window to the file designated by the command. All commands are redirected until you issue the *END command. If an existing file has the same name as the macro file name you specify, the ANSYS program overwrites the existing file.

For example, suppose that you want to create a macro called matprop.mac, which automatically defines a set of material properties. The set of commands entered into the input window for this macro might look like this:

*CREATE,matprop.mac,macros
MP, EX, 1, 2.07E11
MP, NUXY, 1, .27
MP, DENS, 1, 7835
The *CREATE command takes arguments of the file name, the file extension, and the directory path (in this case, the macros directory is specified).

When using *CREATE, all parameters used in commands are written to the file (the currently assigned values for the parameter are not substituted).

You cannot use *CREATE within a DO loop.

**4.1.3.2. Using *CFWRITE**

If you wish to create a macro file in which current values are substituted for parameters you can use *CFWRITE. Unlike *CREATE, the *CFWRITE command cannot specify a macro name; you must first specify the macro file with the *CFOPEN command. Only those ANSYS commands that are explicitly prefaced with a *CFWRITE command are then written to the designated file; all other commands entered in the command input window are executed. As with the *CREATE command, *CFOPEN can specify a file name, a file extension, and a path. The following example writes a BLOCK command to the currently open macro file.

*cfwrite,block,,a,,b,,c

Note that parameters were used for arguments to the BLOCK command. The current value of those parameters (and not the parameter names) are written to the file. So, for this example, the line written to the macro file might be

*cfwrite,block,,4,,2.5,,2

To close the macro file, issue the *CFCLOSE command.

---

**Note**

While it is possible to create a macro through this method, these commands are most useful as a method for writing ANSYS commands to a file during macro execution.

**4.1.3.3. Using /TEE**

Issuing /TEE,NEW or /TEE,APPEND redirects ANSYS commands entered in the command input window to the file designated by the command at the same time that the commands are being executed. All commands are executed and redirected until you issue the /TEE,END command. If an existing file has the same name as the macro file name you specify with /TEE,NEW, the ANSYS program overwrites the existing file. To avoid this, use /TEE,APPEND instead.

In addition to the Label argument (which can have a value of NEW, APPEND, or END), the /TEE command takes arguments of the file name, the file extension, and the directory path.

As the commands are executed in the current ANSYS session, all parameter names are resolved to their current values. However, in the file that is created, parameter names are written (the currently assigned values for the parameter are not substituted). If your current parameter values are important, you can save the parameters to a file using the PARSAV command.

For an example, see the description of the /TEE command in the Command Reference.
4.1.3.4. Using Utility Menu > Macro > Create Macro

Choosing this menu item opens an ANSYS dialog box that you can use as a simple editor for creating macros. You cannot open and edit an existing macro with this facility; if you use the name of an existing macro as the arguments for the *CREATE field, the existing file will be overwritten.

Figure 4.2: The Create Menu Dialog Box

As with the *CREATE command, parameters are not evaluated but are written verbatim into the macro file. Note that you do not make the last line a *END command.

4.1.4. Creating Macros with a Text Editor

You can use your favorite text editor to create or edit macro files. Any ASCII editor will work. Moreover, ANSYS macros can have their lines terminated by either UNIX or Windows line ending conventions (carriage-return, line-feed pairs or simply line-feeds) so you can create a macro on one platform and use it on several platforms.

If you use this method to create macros, do not include the *CREATE and *END commands.
4.1.5. Using Macro Library Files

As a convenience, ANSYS allows you to place a set of macros in a single file, called a macro library file. You can create these either through the *CREATE command or through a text editor. Given that macro libraries tend to be longer than single macros, using a text editor normally provides the best approach.

Macros libraries have no explicit file extension and follow the same file naming conventions as macro files. A macro library file has the following structure:

MACRONAME1
.
.
./EOF
MACRONAME2
.
.
./EOF
MACRONAME3
.
.
./EOF

For example, the following macro file contains two simple macros:

mybloc
/prep7
/view,,-1,-2,-3
block,,4,,3,,2
finish
./EOF
mysphere
/prep7
/view,,-1,-2,-3
sphere,1
finish
./EOF

Note that each macro is prefaced with a macro name (sometimes referred to as a data block name) and ends with a /EOF command.
A macro library file can reside anywhere on your system, although for convenience you should place it within the macro search path. Unlike macro files, a macro library file can have any extension up to eight characters.

### 4.2. Executing Macros and Macro Libraries

You can execute any macro file by issuing the *USE command. For example, to execute the macro called MYMACRO (no extension) residing in the current working directory, you would issue

```
*use,mymacro
```

In this case, the macro takes no arguments. If instead the macro was called MYMACRO.MACRO and resided in /myaccount/macros, you could call it with

```
*use,/myaccount/macros/mymacro.macro
```

Note that the *USE command allows you to enter the path and extension along with the file name and that these are not entered as separate arguments.

If a macro has a .mac file extension and resides in the search path, you can execute it as if it were an ANSYS command by simply entering it in the command input window. For example, to call mymacro.mac you could simply enter

```
mymacro
```

You can also execute macros with a .mac extension through the Utility Menu> Macro> Execute Macro menu item.

If the same macro takes arguments (see Passing Arguments to a Macro (p. 52) for more information about passing arguments to macros), then these can be entered on the command line as follows

```
mymacro,4,3,2,1.5
```

or

```
*use,mymacro.mac,4,3,2,1.5
```

The Utility Menu> Macro> Execute Macro menu item dialog provides fields for arguments.

Executing macros contained in macro libraries is similar. You must first specify the library file using the *ULIB command. For example, to specify that macros are in the mymacros.mlib file, which resides in the /myaccount/macros directory, you would issue the following command:

```
*ulib,mymacros.mlib,/myaccount/macros/
```

After selecting a macro library, you can execute any macro contained in the library by specifying it through the *USE command. As with macros contained in individual files, you can specify arguments as parameters in the *USE command.

---

**Note**

You cannot use the *USE command to access macros not contained in the specified macro library file after issuing the *ULIB command.
4.3. Local Variables

APDL provides two sets of specially named scalar parameters which are available for use as local variables. These consist of

- A set of scalar parameters that provide a way of passing command line arguments to the macro.
- A set of scalar parameters that can be used within the macro. These provide a set of local variables that can be used to define values only within that macro.

The following sections discuss both of these variable types in detail.

4.3.1. Passing Arguments to a Macro

There are 19 scalar parameters that you can use to pass arguments from the macro execution command line to the macro. These scalar parameters can be reused with multiple macros; that is, their values are local to each macro. The parameters are named ARG1 through AR19 and they can be used for any of the following items:

- Numbers
- Alphanumeric character strings (up to eight characters enclosed in single quotes)
- Numeric or character parameters
- Parametric expressions

Note

You can pass only the values of parameters ARG1 through AR18 to a macro as arguments with the *USE command. If you create a macro that can be used as an ANSYS command (the macro files has a .mac extension), you can pass the values of parameters ARG1 through AR19 to the macro.

For example, the following simple macro requires four arguments, ARG1, ARG2, ARG3, and ARG4:

```
/prep7
/view,-1,-2,-3
block,,arg1,,arg2,,arg3
sphere,arg4
vsbv,1,2
finish
```

To execute this macro, a user might enter

```
mymacro,4,3,2,2,1
```

4.3.2. Local Variables Within Macros

Each macro can have up to 80 scalar parameters used as local variables (AR20 through AR99). These parameters are completely local to the macro, and multiple macros can each have their own unique values assigned to these parameters. These parameters are not passed to macros called from macros (nested macros). They are passed to any files processed through a /INPUT command or a "do loop" processed within the macro.
4.3.3. Local Variables Outside of Macros

ANSYS also has a similar set of ARG1 through AR99 scalar parameters that are local to an input file, and are not passed to any macros called by that input file. Thus, once a macro finishes and execution returns to an input file, the values of ARG1 through AR99 revert to whatever values were defined within the input file.

4.4. Controlling Program Flow in APDL

When executing an input file, ANSYS is normally restricted to linear program flow; that is, each statement is executed in the order that it is encountered in the listing. However, APDL provides a rich set of commands that you can use to control program flow. These commands are listed in Chapter 2 of the Command Reference.

- Call subroutines (nested macros).
- Branch unconditionally to a specified location with a macro.
- Branch based upon a condition to a specified location within a macro.
- Repeat the execution of a single command, incrementing one or more command parameters.
- Loop through a section of a macro a specified number of times.

The following sections detail each of these program control capabilities. For the exact syntax of the commands, refer to the Command Reference.

4.4.1. Nested Macros: Calling Subroutines Within a Macro

4.4.2. Unconditional Branching: Goto

4.4.3. Conditional Branching: The *IF Command

4.4.4. Repeating a Command

4.4.5. Looping: Do-Loops

4.4.6. Implied (colon) Do Loops

4.4.7. Additional Looping: Do-While

4.4.1. Nested Macros: Calling Subroutines Within a Macro

APDL allows you to nest macros up to 20 levels deep, providing functionally similar capability to a FORTRAN CALL statement or to a function call. You can pass up to 19 arguments to the macro and, at the conclusion of each nested macro, execution returns to the level that called the macro. For example, the following simply macro library file shows the MYSTART macro, which calls the MYSFHERE macro to create the sphere.

```plaintext
mystart
/prep7
/view,-1,-2,-3
mysphere,1.2
finish
/eof
mysphere
sphere,arg1
/eof
```

4.4.2. Unconditional Branching: Goto

The simplest branching command, *GO, instructs the program to go to a specified label without executing any commands in between. Program flow continues from the specified label. For example

```plaintext
*GO,:BRANCH1
--- ! This block of commands is skipped (not executed)
---
:BRANCH1
---
```
The label specified by the \texttt{*GO} command must start with a colon (:) and must not contain more than eight characters, including the colon. The label can reside anywhere within the same file.

\textbf{Note}

The use of \texttt{*GO} is now considered obsolete and is discouraged. See the other branching commands for better methods of controlling program flow.

### 4.4.3. Conditional Branching: The \texttt{*IF} Command

APDL allows you to execute one of a set of alternative blocks based on the evaluation of a condition. The conditions are evaluated by comparing two numerical values (or parameters that evaluate to numerical values).

The \texttt{*IF} command has the following syntax

\texttt{*IF, VAL1, Oper, VAL2, Base}

Where

- \textit{VAL1} is the first numerical value (or numerical parameter) in the comparison.
- \textit{Oper} is the comparison operator.
- \textit{VAL2} is the second numerical value (or numerical parameter) in the comparison.
- \textit{Base} is the action that occurs if the comparison evaluates as true.

APDL offers eight comparison operators, which are discussed in detail in the \texttt{*IF} command reference. Briefly these are:

\begin{itemize}
  \item \texttt{EQ} \quad Equal (for \textit{VAL1} = \textit{VAL2}).
  \item \texttt{NE} \quad Not equal (for \textit{VAL1} \neq \textit{VAL2}).
  \item \texttt{LT} \quad Less than (for \textit{VAL1} < \textit{VAL2}).
  \item \texttt{GT} \quad Greater than (for \textit{VAL1} > \textit{VAL2}).
  \item \texttt{LE} \quad Less than or equal (for \textit{VAL1} \leq \textit{VAL2}).
  \item \texttt{GE} \quad Greater than or equal (for \textit{VAL1} \geq \textit{VAL2}).
  \item \texttt{ABLT} \quad Absolute values of \textit{VAL1} and \textit{VAL2} before < operation.
  \item \texttt{ABGT} \quad Absolute values of \textit{VAL1} and \textit{VAL2} before > operation.
\end{itemize}

By giving the \textit{Base} argument a value of THEN, the \texttt{*IF} command becomes the beginning of an if-then-else construct (similar to the FORTRAN equivalent). The construct consists of

- An \texttt{*IF} command, followed by
4.4.3. Conditional Branching: The *IF Command

- One or more optional *ELSEIF commands
- An optional *ELSE command
- A required *ENDIF command, marking the end of the construct.

In its simplest form, the *IF command evaluates the comparison and, if true, branches to a label specified in the Base argument. This is similar to the "computed goto" in FORTRAN. (In combination, a set of such *IF commands could function similarly to the CASE statements in other programming languages.) Take care not to branch to a label within an if-then-else construct or do-loop. If a batch input stream hits an end-of-file during a false *IF condition, the ANSYS run will not terminate normally. You will need to terminate it externally (use either the UNIX “kill” function or the Windows task manager).

By setting the Base argument to a value of STOP, you can exit from ANSYS based on a particular condition.

An if-then-else construct simply evaluates a condition and executes the following block or jumps to the next statement following the *ENDIF command (shown with the "Continue" comment).

```
*IF,A,EQ,1,THEN
   ! Block1
   .
*ENDIF
   ! Continue
```

The following example shows a more complex structure. Note that only one block can be executed. If no comparison evaluates to true, the block following the *ELSE command is executed.

**Figure 4.4: A Sample If-Then-Else Construct**

```
*IF,A,EQ,1,THEN
   ! Block1
   .
   *ELSEIF,A,EQ,0
   ! Block2
   .
   *ELSEIF,A,EQ,3
   ! Block3
   .
   *ELSE
   ! Block4
   .
*ENDIF
   ! Continue
```

**Note**

You can issue a /CLEAR command within an if-then-else construct. The /CLEAR command does not clear the *IF stack and the number of *IF levels is retained. An *ENDIF is necessary to close any branching logic. Also, keep in mind that the /CLEAR command deletes all parameters, including any that are used in your branching commands. You can avoid any problems that might arise from the deletion of parameters by issuing a PARSAV command before the /CLEAR command, and then following the /CLEAR command with a PARRES command.
4.4.4. Repeating a Command

The simplest looping capability, the *REPEAT command, allows you to execute the directly preceding command a specified number of times, incrementing any field in that command by a constant value. In the example

E,1,2
*REPEAT,5,0,1

the E command generates one element between nodes 1 and 2 and the following *REPEAT command specifies that E executes a total of five times (including the original E command), incrementing the second node number by one for each additional execution. The result is five total elements with node connectivities 1-2, 1-3, 1-4, 1-5, and 1-6.

Note

Most commands that begin with a slash (/) or an asterisk (*), as well as macros executed as "unknown commands," cannot be repeated. However, graphics commands that begin with a slash can be repeated. Also, avoid using the *REPEAT command with interactive commands, such as those that require picking or those that require a user response.

4.4.5. Looping: Do-Loops

A do-loop allows you to loop through a series of commands a specified number of times. The *DO and *ENDDO commands mark the beginning and ending points for the loop. *DO command has the following syntax:

The following example do-loop edits five load step files (numbered 1 through 5) and makes the same changes in each file.

*DO,I,1,5       ! For I = 1 to 5:
LSREAD,I        ! Read load step file I
OUTPR,ALL,NONE  ! Change output controls
ERESX,NO
LSWRITE,I       ! Rewrite load step file I
*ENDDO

You can add your own loop controls by using the *IF, *EXIT, or *CYCLE commands.

Keep the following guidelines in mind when constructing do-loops.

- Do not branch out of a do-loop with a :Label on the *IF or *GO commands.
- Avoid using a :Label to branch to a different line within a do-loop. Use if-then-else-endif instead.
- Output from commands within a do-loop is automatically suppressed after the first loop. Use /GOPR or /GO (no response line) within the do-loop if you need to see output for all loops.
- Take care if you include a /CLEAR command within a do-loop. The /CLEAR command does not clear the do-loop stack, but it does clear all parameters including the loop parameter in the *DO statement itself. You can avoid the problem of having an undefined looping value by issuing a PARSAV command before the /CLEAR command, and then following the /CLEAR command with a PARRES command.

4.4.6. Implied (colon) Do Loops

You can also use the implied (colon) convention for do loops. Using this convention is typically faster because the looping is done in memory. The correct syntax is:
with \( z \) defaulting to 1 if not specified. For example:

\[ n, (1:6), (2:12:2) \]

will perform the same steps as:

\[ n, 1, 2 \]
\[ n, 2, 4 \]
\[ n, 3, 6 \]
\[ \ldots \]
\[ n, 6, 12 \]

When using the implied (colon) do loops, be aware that the shortest expression controls execution. For example,

\[ n, (1:7), (2:12:2) \]

would behave identically to the example above.

Additional numeric fields that do not have the colon (:) will be taken as a constant value.

Also, non-integer numbers will function normally. However, if non-integer numbers are applied to a command that requires integers, then the non-integer will be rounded off following normal mathematical conventions.

This looping convention can be used only for fields requiring a numeric entry. Looping may also be used with GET function arguments, for example \( a(1:5)=nx(1:5) \). A text entry field will process \((x:y:z)\) as a literal value.

### 4.4.7. Additional Looping: Do-While

You can also perform looping functions that will repeat indefinitely until an external parameter changes. The *DOWHILE command has the following syntax:

*\text{DOWHILE}, Parm*

The loop repeats as long as the parameter \( Parm \) is TRUE. If \( Parm \) becomes false (less than or equal to 0.0), the loop terminates. The *CYCLE and *EXIT commands can be used within a *DOWHILE loop.

### 4.5. Control Functions Quick Reference

The table below describes APDL commands that perform control functions within macros.

Most of the important information about these commands appears here, but you may want to look at the complete command descriptions in the *Command Reference*.

<table>
<thead>
<tr>
<th>APDL Command</th>
<th>Action It Takes</th>
<th>Usage Tips</th>
</tr>
</thead>
</table>
| *DO*         | Defines the start of a "do" loop. The commands following the *DO command execute (up to the *EN-DDO command) re- | • You can also control looping via the *IF command.  
• ANSYS allows up to 20 levels of nested "do" loops, although "do" loops that include /INPUT, *USE, or an “unknown” command macro support fewer nesting levels because they do internal file switching. |
<table>
<thead>
<tr>
<th>APDL Command</th>
<th>Action It Takes</th>
<th>Usage Tips</th>
</tr>
</thead>
</table>
| *DO, *ENDDO, *CYCLE, and *EXIT | Peatedly until some loop control is satisfied. | • *DO, *ENDDO, *CYCLE, and *EXIT commands in a “do” loop must all read from the same file or the keyboard.  
• *DO not include picking operations in a “do” loop.  
• Be careful if you include a /CLEAR command within a do-loop. The /CLEAR command does not clear the do-loop stack, but it does clear all parameters including the loop parameter in the *DO statement itself. You can avoid the problem of having an undefined looping value by issuing a PARSAV command before the /CLEAR command, and then following the /CLEAR command with a PARRES command. |

| *ENDDO | Ends a “do” loop and starts the looping action. | You must use one *ENDDO command for each nested “do” loop. The *ENDDO and *DO commands for a loop must be on the same file.  
You can use the cycle option conditionally (via the *IF command). The *CYCLE command must appear on the same file as the *DO command and must appear before the *ENDDO command. |

| *CYCLE | When executing a “do” loop, ANSYS bypasses all commands between the *CYCLE and *ENDDO commands, then (if applicable) initiates the next loop. | You can use the cycle option conditionally (via the *IF command). The *CYCLE command must appear on the same file as the *DO command and must appear before the *ENDDO command. |

| *EXIT | Exits from a “do” loop. | The command following the *ENDDO command executes next. The *EXIT and *DO commands for a loop must be on the same file. You can use the exit option conditionally (via the *IF command). |

| *IF | Causes commands to be read conditionally. | • You can have up to 10 nested levels of *IF blocks.  
• You cannot jump into, out of, or within a “do” loop or an if-then-else construct to a :label line, and jumping to a :label line is not allowed with keyboard entry.  
• You can issue a /CLEAR command within an if-then-else construct. The /CLEAR command does not clear the *IF stack and the number of *IF levels is retained. An *ENDIF is necessary to close any branching logic.  
• The /CLEAR command deletes all parameters, including any that are used in your branching commands. You can avoid any problems that might arise from the deletion of parameters by issuing a PARSAV command before the /CLEAR command, and then following the /CLEAR command with a PARRES command. |

| *ENDIF | Terminates an if-then-else construct. (See the *IF discussion for details.) | The *IF and *ENDIF commands must appear in the same file. |
**APDL Command** | **Action It Takes** | **Usage Tips**
--- | --- | ---
*ELSE* | Creates a final, optional block separator within an if-then-else construct. (See the *IF* discussion for details.) | The *ELSE* and *IF* commands must appear in the same file.

*ELSEIF* | Creates an optional, intermediate block separator within an if-then-else construct. | If Oper = EQ or NE, VAL1 and VAL2 can also be character strings (enclosed in quotes) or parameters. The *IF* and *ELSEIF* commands must be on the same file.

---

### 4.6. Using the _STATUS and _RETURN Parameters in Macros

The ANSYS program generates two parameters, _STATUS and _RETURN, that you can also use in your macros. For example, you might use the _STATUS or _RETURN value in an "if-then-else" construct to have the macro take some action based on the outcome of executing an ANSYS command or function.

Solid modeling functions generate the _RETURN parameter, which contains the result of executing the function. The following table defines the _RETURN values for the various solid modeling functions:

#### Table 4.1 _RETURN Values

<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
<th>_RETURN Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Keypoints</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>Defines a keypoint</td>
<td>keypoint number</td>
</tr>
<tr>
<td>KL</td>
<td>Keypoint on a line</td>
<td>Keypoint number</td>
</tr>
<tr>
<td>KNODE</td>
<td>Keypoint at node</td>
<td>Keypoint number</td>
</tr>
<tr>
<td>KBETW</td>
<td>Keypoint between two keypoints</td>
<td>KP number</td>
</tr>
<tr>
<td>KCENTER</td>
<td>Keypoint at center</td>
<td>KP number</td>
</tr>
<tr>
<td><strong>Lines</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSPLIN</td>
<td>Generate spline</td>
<td>Line number</td>
</tr>
<tr>
<td>CIRCLE</td>
<td>Generate circular arc lines</td>
<td>First line number</td>
</tr>
<tr>
<td>L</td>
<td>Line between two keypoints</td>
<td>Line number</td>
</tr>
<tr>
<td>L2ANG</td>
<td>Line at angle with two lines</td>
<td>Line number</td>
</tr>
<tr>
<td>LANG</td>
<td>Line tangent to two lines</td>
<td>Line number</td>
</tr>
<tr>
<td>LARC</td>
<td>Defines a circular arc</td>
<td>Line number</td>
</tr>
<tr>
<td>LAREA</td>
<td>Line between two keypoints</td>
<td>Line number</td>
</tr>
<tr>
<td>LCOMB</td>
<td>Combine two lines into one</td>
<td>Line number</td>
</tr>
<tr>
<td>LDIV</td>
<td>Divide line into two or more lines</td>
<td>First keypoint number</td>
</tr>
<tr>
<td>LDRAG</td>
<td>Line by keypoint sweep</td>
<td>First line number</td>
</tr>
<tr>
<td>LFILLT</td>
<td>Fillet line between two lines</td>
<td>Fillet line number</td>
</tr>
<tr>
<td>LROTAT</td>
<td>Arc by keypoint rotation</td>
<td>First line number</td>
</tr>
<tr>
<td>LSTR</td>
<td>Straight line</td>
<td>Line number</td>
</tr>
<tr>
<td>Command</td>
<td>Function</td>
<td>_RETURN Value</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>---------------</td>
</tr>
<tr>
<td>LTAN</td>
<td>Line at end and tangent</td>
<td>Line number</td>
</tr>
<tr>
<td>SPLINE</td>
<td>Segmented spline</td>
<td>First line number</td>
</tr>
</tbody>
</table>

**Areas**

<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
<th>_RETURN Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Area connecting keypoints</td>
<td>Area number</td>
</tr>
<tr>
<td>ACCAT</td>
<td>Concatenate two or more areas</td>
<td>Area number</td>
</tr>
<tr>
<td>ADRAG</td>
<td>Drag lines along path</td>
<td>First area number</td>
</tr>
<tr>
<td>AFILLT</td>
<td>Fillet at intersection of two areas</td>
<td>Fillet area number</td>
</tr>
<tr>
<td>AL</td>
<td>Area bounded by lines</td>
<td>Area number</td>
</tr>
<tr>
<td>ALPFILL</td>
<td>All loops</td>
<td>Area number</td>
</tr>
<tr>
<td>AOFFST</td>
<td>Area offset from given area</td>
<td>Area number</td>
</tr>
<tr>
<td>AROTAT</td>
<td>Rotate lines around axis</td>
<td>First area number</td>
</tr>
<tr>
<td>ASKIN</td>
<td>Skin surface through guiding lines</td>
<td>First area number</td>
</tr>
<tr>
<td>ASUB</td>
<td>Area using shape of existing area</td>
<td>Area number</td>
</tr>
</tbody>
</table>

**Volumes**

<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
<th>_RETURN Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>Volume through keypoints</td>
<td>Volume number</td>
</tr>
<tr>
<td>VA</td>
<td>Volume bounded through areas</td>
<td>Volume number</td>
</tr>
<tr>
<td>VDRAG</td>
<td>Drag area pattern to create volume</td>
<td>First volume number</td>
</tr>
<tr>
<td>VEXT</td>
<td>Volume by extruding areas</td>
<td>First volume number</td>
</tr>
<tr>
<td>VOFFST</td>
<td>Volume offset from given area</td>
<td>Volume number</td>
</tr>
<tr>
<td>VROAT</td>
<td>Volume by rotating areas</td>
<td>First volume number</td>
</tr>
</tbody>
</table>

Executing an ANSYS command, whether in a macro or elsewhere, generates the parameter _STATUS. This parameter reflects the error status of that command:

- 0 for no error
- 1 for a note
- 2 for a warning
- 3 for an error

### 4.7. Using Macros with Components and Assemblies

To make large models easier to manage, you may want to divide a model into discrete components based on different types of entities: nodes, elements, keypoints, lines, areas, or volumes. Each component can contain only one type of entity. Doing this enables you to perform tasks such as applying loads or producing graphics displays conveniently and separately on different portions of the model.

You can also create assemblies, which are groups that combine two or more components or even multiple assemblies. You can nest assemblies up to five levels deep. For example, you could build an assembly named motor from components called STATOR, PERMMAG, ROTOR, and WINDINGS.

The table below describes some of the commands you can issue to build components and assemblies. For more detailed discussions of these commands, see the Command Reference. For further information on components and assemblies, see Selecting and Components in the Basic Analysis Guide.
CM Groups geometry items into a component
CMDELE Deletes a component or assembly.
CMEDIT Edits an existing component or assembly. ANSYS updates assemblies automatically to reflect deletions of lower-level or assemblies.
CMGRP Groups components and assemblies into one assembly. Once defined, an assembly can be listed, deleted, selected, or unselected using the same commands as for components.
CMLIST Lists the entities contained in a component or assembly.
CMSEL Selects a subset of components and assemblies.

4.8. Reviewing Example Macros

Following are two example macros. The example macro below, called offset.mac, offsets selected nodes in the PREP7 preprocessor. This macro is for demonstration purposes only because the NGEN command provides a more convenient method.

```ansys
! Macro to offset selected nodes in PREP7
! The below file is saved as: offset.mac (must be lowercase)
! Usage: offset,dx,dy,dz
/nop            ! suppress printout for this macro
*get,nnode,node,,num,max   ! get number of nodes
*dim,x,,nnode       ! set up arrays for node locations
*dim,y,,nnode
*dim,z,,nnode
*dim,sel,,nnode    ! set up array for select vector
*dim,loc,,nnode
*get,x(1),node,1,loc,x   ! get coordinates
*get,y(1),node,1,loc,y
*get,z(1),node,1,loc,z
*get,sel(1),node,1,nsel  ! get selected set
*voper,x(1),x(1),add,arg1 ! offset locations
*voper,y(1),y(1),add,arg2
*voper,z(1),z(1),add,arg3
! *do,i,1,nnode           ! store new positions
!  *if,sel(i),gt,0,then   ! this form takes 98 sec for 100,000 nodes
!    n,i,x(i),y(i),z(i)
!  *endif
! *enddo
*vmask,sel(1)         ! takes 3 seconds for 100,000 nodes
n,(1:NNODE),x(1:NNODE),y(1:NNODE),z(1:NNODE)
   x(1) =                   ! delete parameters (cleanup)
y(1) =
z(1) =
   sel(1) =
i=
nnode=
/go            ! resume printout
```

The following example macro, called bilinear.mac, evaluates two bilinear materials. This is a useful macro that can be run after solving a static analysis. Material 1 is the tension properties, and Material 2 is the compression properties. ARG1 is the number of iterations (default is 2).
nop

_niter = arg1                      ! set number of iterations
*if,_niter,lt,2,then
  _Niter = 2
endif
*do,iter,1,_niter                  ! loop on number of iterations
/post1
set,1,1
*get,ar11,elem,,num,maxd
*dim,_s1,,ar11                     ! array for element s1
*dim,_s3,,ar11                     ! array for element s3
etable,sigmax,s,1
etab,sigmin,s,3
*get,_s1(1),elem,1,etab,sigmax     ! get element maximum stress in s1
*get,_s3(1),elem,1,etab,sigmin     ! get element minimum stress in s3
*dim,_mask,,ar11                   ! array for mask vector
*vget,_mask(1),_s1(1),lt,0         ! true if max. stress < 0
*vcum,1                           ! accumulate compression elements
*vabs,0,1                          ! absolute value of s3
*voper,_mask(1),_s3(1),gt,_s1(1)   ! true if abs(minstr) > maxstr
finish

/prep7                             ! go to prep7 for element material mods
mat,1                              ! set all materials to tension properties
emod,all
*vput,_mask(1),elem,1,esel         ! select compression elements
mat,2                              ! change selected elements to compression
emod,all

call                               ! select all elements
finish

/_s1(1)=                           ! clean up all vectors (set to zero)
/_s3(1)=
/_mask(1)=
/solve solve
finish
*endo do                             ! end of iterations

/_niter=
/_iter=
/gop
Within an ANSYS macro, you have several ways to access components of the ANSYS graphical user interface (GUI):

- You can modify and update the ANSYS toolbar (this is discussed in detail in Adding Commands to the Toolbar).
- You can issue the *ASK command to prompt a user to enter a single parameter value.
- You can create a dialog box to prompt a user to enter multiple parameter values.
- You can issue the *MSG command to have the macro write an output message.
- You can have the macro update or remove a status bar.
- You can allow the user to select entities through graphical picking from within a macro.
- You can call any dialog box.

The following GUI topics are available:

5.1. Prompting Users for a Single Parameter Value
5.2. Prompting Users With a Dialog Box
5.3. Using Macros to Display Your Own Messages
5.4. Creating and Maintaining a Status Bar from a Macro
5.5. Picking within Macros
5.6. Calling Dialog Boxes From a Macro

### 5.1. Prompting Users for a Single Parameter Value

By including the *ASK command within a macro, you can have the macro prompt a user to type in a parameter value.

The format for the *ASK command is

*ASK,Par,Query,DVAL

Where

- *Par* is an alphanumeric name that identifies the scalar parameter used to store the user input.
- *Query* is the text string that ANSYS displays to prompt the user. This string can contain up to 54 characters. *Don't use characters that have special meanings, such as "$" or "!".*
- *DVAL* is the default value given the parameter if a user issues a blank response. This value can be either a one-to-eight character string (enclosed in single quotes) or a number. If you assign no default, a blank user response deletes the parameter.

The *ASK command prints the *Query text on the screen and waits for a response. It reads the response from the keyboard except when ANSYS runs in batch mode. (In that case, the response or responses must be the next-read input line or lines.) The response can be a number, a one-to-eight character string enclosed in single quotes, a numeric or character parameter, or an expression that evaluates to a number. ANSYS
then sets the value of Par to the read-in response. The following example displays the dialog box shown below, then sets the parameter PARM1 to the value the user enters.

*ask,parm1,'username (enclose the username in single quotes)'

**Figure 5.1: An Example *ASK Dialog Box**

When you issue *ASK within a macro, ANSYS writes the user’s response to File.LOG on the line following the macro name.

### 5.2. Prompting Users With a Dialog Box

The MULTIPRO command constructs a simple, multiple-prompt dialog box that can contain up to 10 parameter prompts. The command allows you to use a set of UIDL *CSET commands to create the prompts as well as specify a default value for each prompt. Be aware that macros using MULTIPRO cannot be called from UIDL. You cannot use MULTIPRO within a DO loop.

The MULTIPRO command must be used in conjunction with:

- Between one and ten *CSET command prompts
- Up to two special *CSET commands that provide a two line area for user instructions.

The command has the following syntax:

```
MULTIPRO,'start',Prompt_Num
*CSET,Strt_Loc,End_Loc,Param_Name,'Prompt_String',Def_Value
MULTIPRO,'end'
```

Where

- 'start'
  
  A literal string that, when encountered as the first argument, marks the beginning of the MULTIPRO construct. The literal must be enclosed in single quotes.

- **Prompt_Num**
  
  Required only if Def_Value is omitted from at least one *CSET command or if Def_Value is set to 0. The Prompt_Num value is an integer equal to the number of following *CSET prompts.

- **Strt_Loc,End_Loc**
  
  The initial value for Strt_Loc for the first *CSET command is 1, and the value for End_Loc is Strt_Loc+2 (3 for the first *CSET command). The value of each subsequent Strt_Loc is the previous End_Loc+1.

- **Param_Name**
  
  The name of the parameter that will hold either the value specified by the user or, if the user supplies no value, the value of Def_Value.

- **'Prompt_String'**
  
  A string, which can contain up to 32 characters, which can be used to describe the parameter. This string must be enclosed in single quotes.
Def_Value

Default value used if no value specified by user. Default value can be a numeric expression including APDL numeric parameters. Character expressions are not allowed.

'end'

A literal string, used as the first argument for the closing MULTIPRO command.

The following is a typical example of the MULTIPRO command.

```plaintext
multipro,'start',3
   *cset,1,3,beamW,'Enter the overall beam width',12.5
   *cset,4,6,beamH,'Enter the beam height',23.345
   *cset,7,9,beamL,'Enter the beam length',50.0
multipro,'end'
```

Up to two optional *CSET commands can be added to the construct that can provide two 64 character strings. You can use these to provide instructions to the user. The syntax for these specialized *CSET commands is

```plaintext
*CSET,61,62,'Help_String','Help_String'*CSET,63,64,'Help_String','Help_String'
```

Where

'Help_String'

A string which can contain up to 32 characters. If you need more than 32 characters, you can use a second Help_String argument.

The following is an example of a MULTIPRO construct using the optional help lines. Note that two Help_String arguments are used to overcome the 32 character limit.

```plaintext
multipro,'start',3
   *cset,1,3,dx,'Enter DX Value',0.0
   *cset,4,6,dy,'Enter DY Value',0.0
   *cset,7,9,dz,'Enter DZ Value',0.0
   *cset,61,62,'The MYOFSET macro offsets the',' selected nodes along each'
      *cset,63,64,'of the three axes. Fill in the ',' fields accordingly.'
multipro,'end'
```

The above construct creates the following multiple-prompt dialog box.

**Figure 5.2: A Typical Multiple-Prompt Dialog Box**

You can check the status of the buttons by testing the value of the _BUTTON parameter. The following lists the button status values:

- _BUTTON = 0 indicates that the OK button was pressed.
5.3. Using Macros to Display Your Own Messages

By issuing the *MSG command within a macro, you can display custom output messages via the ANSYS message subroutine. The command has the following format:

*MSG, Lab, VAL1, VAL2, VAL3, VAL4, VAL5, VAL6, VAL7, VAL8

Where Lab is one of the following labels for output and termination control:

<table>
<thead>
<tr>
<th>Lab</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFO</td>
<td>Writes the message with no heading (default).</td>
</tr>
<tr>
<td>NOTE</td>
<td>Writes the message with a &quot;NOTE&quot; heading.</td>
</tr>
<tr>
<td>WARN</td>
<td>Writes the message with a &quot;WARNING&quot; heading, and also writes it to the errors file, Jobname.ERR.</td>
</tr>
<tr>
<td>ERROR</td>
<td>Writes the message with an &quot;ERROR&quot; heading and also writes it to the errors file, Jobname.ERR. If this is an ANSYS batch run, this label also terminates the run at the earliest &quot;clean exit&quot; point.</td>
</tr>
<tr>
<td>FATAL</td>
<td>Writes the message with a &quot;FATAL ERROR&quot; heading and also writes it to the errors file, Jobname.ERR. This label also terminates the ANSYS run immediately.</td>
</tr>
<tr>
<td>UI</td>
<td>Writes the message with a &quot;NOTE&quot; heading and displays it in the message dialog box.</td>
</tr>
</tbody>
</table>

VAL1 through VAL8 are numeric or alphanumeric character values to be included in the message. Values can be the results of evaluating parameters. All numeric values are assumed to be double precision.

You must specify the message format immediately after the *MSG command. The message format can contain up to 80 characters, consisting of text strings and predefined "data descriptors" between the strings where numeric or alphanumeric character data are to be inserted. These data descriptors are:

- %i, for integer data. The FORTRAN nearest integer (NINT) function is used to form integers for the %i descriptor.
- %g, for double precision data
- %c, for alphanumeric character data
- %/, for a line break

The corresponding FORTRAN data descriptors for the first three descriptors are I9, 1PG16.9, and A8 respectively. A blank must precede each descriptor. You also must supply one data descriptor for each specified value (eight maximum), in the order of the specified values.

Don't begin *MSG format lines with *IF, *ENDIF, *ELSE, or *ELSEIF. If the last non-blank character of the message format is an ampersand (&), the ANSYS program reads a second line as a continuation of the format. You can use up to 10 lines (including the first) to specify the format information.

Consecutive blanks are condensed into one blank upon output, and a period is appended. The output produced can be up to 10 lines of 72 characters each (using the %/ descriptor).

The example below shows you an example of using *MSG that prints a message with two integer values and one real value:

*MSG, INFO, 4Inner4,25,1.2,148
Radius (%C) = %I, Thick = %G, Length = %I
The resulting output message is as follows:

Radius (Inner) = 25, Thick = 1.2, Length = 148

Here is an example illustrating multiline displays in GUI message windows:

*MSG,UI,Vcoilrms,THTAv,Icoilrms,THTAi,Pappnt,Pelec,PF,indctnc
Coil RMS voltage, RMS current, apparent pwr, actual pwr, pwr factor: %/&
Vcoil = %G V (electrical angle = %G DEG) %/&
Icoil = %G A (electrical angle = %G DEG) %/&
APPARENT POWER = %G W %/&
ACTUAL POWER = %G W %/&
Power factor: %G %/&
Inductance = %G %/&
VALUES ARE FOR ENTIRE COIL (NOT JUST THE MODELED SECTOR)

---

Note

The command /UIS,MSGPOP controls which messages a message dialog box displays when the GUI is active. See the Command Reference for more information about this command.

5.4. Creating and Maintaining a Status Bar from a Macro

Within macros, you can insert commands to define an ANSYS dialog box containing a status bar displaying the progress of an operation, a STOP button you can click on to stop the operation, or both.

To define a status dialog box, issue the following command:

```
*ABSET,Title40,Item
```

- **Title40** is the text string that appears in the dialog box with the status bar. The string can contain a maximum of 40 characters.
- **Item** is one of the following values:
  - **BAR** Displays the status bar with no STOP button
  - **KILL** Displays a STOP button with no status bar
  - **BOTH** Displays both the status bar and STOP button

To update the status bar, issue the command `*ABCHECK,Percent,NewTitle`.

- **Percent** is an integer between 0 and 100. It gives the position of the status bar.
- **NewTitle** is a 40-character string that contains progress information. If you specify a string for `NewTitle`, it replaces the string supplied in `Title40`.

If you specify KILL or BOTH, your macro should check the _RETURN parameter after each execution of `*ABCHECK` to see if the user has pressed the STOP button, then take the appropriate action.

To remove the status bar from the ANSYS GUI, issue the `*ABFINI` command.

The following example macro illustrates the status bar (complete with bar and STOP button) in use. The status dialog box that is produced is shown in the following figure. Note that the macro checks the status of the _RETURN parameter and, if the STOP button is pressed, posts the “We are stopped......” message.

```
fini
/clear,nost
/prep7
```
n,1,1
n,1000,1000
fill
*abset,'This is a Status Bar',BOTH
myparam = 0
*do,i,1,20
  j = 5*i
  *abcheck,j
  *if,_return,gt,0,then
    myparam = 1
  *endif
  *if,myparam,gt,0,exit
  /ang,,j
  nplot,1
  *if,_return,gt,0,then
    myparam = 1
  *endif
  *if,myparam,gt,0,exit
  nlist,all
  *if,_return,gt,0,then
    myparam = 1
  *endif
  *if,myparam,gt,0,exit
*endo
*if,myparam,gt,0,then
*msg,ui
  We are stopped..........
*endif
*abfinish
fini

Note

Do not call *ABCHECK more than about 20 times in a loop.

Figure 5.3: A Typical Status Dialog Box

5.5. Picking within Macros

If you’re running the ANSYS program interactively, you can call a GUI picking menu from within a macro. To do so, simply include a picking command in the macro. Many ANSYS commands (such as K,P) accept the input "P" to enable graphical picking. When ANSYS encounters such a command, it displays the appropriate picking dialog and then continues macro execution when the user clicks OK or Cancel.

Keep in mind that picking commands are not available in all ANSYS processors, and that you must first switch to an appropriate processor before calling the command.

Note

If a macro includes GUI functions, the /PMACRO command should be the first command in that macro. This command causes the macro contents to be written to the session log file. This is important, because if you omit the /PMACRO command, ANSYS can’t read the session log file to reproduce the ANSYS session.
5.6. Calling Dialog Boxes From a Macro

When the ANSYS program encounters a dialog box UIDL function name (such as Fnc_UIMP_Is0), it displays the appropriate dialog box. Thus, you can launch any ANSYS dialog box by making its function name a separate line in the macro file. When you dismiss that dialog box, the program continues processing the macro starting with the next line after the function call.

Keep in mind that many dialog boxes have a number of dependencies, including that the appropriate ANSYS processor is active and that certain required preexisting conditions are met. For example, launching a dialog box to select nodes first supposes that nodes exist, if no nodes exist the macro will fail when the user clicks OK or Apply.

---

**Note**

If a macro includes GUI functions, the `/PMACRO` command should be the first command in that macro. This command causes the macro contents to be written to the session log file. This is important, because if you omit the `/PMACRO` command, ANSYS can't read the session log file to reproduce the ANSYS session.
Chapter 6: Encrypting Macros

ANSYS provides the ability to encrypt macro files so that the source is not "human-readable." Encrypted macros require an encryption key to run. You can either place the encryption key explicitly (in readable ASCII) in the macro or you can set it in ANSYS as a global encryption key.

The following macro encryption topics are available:

6.1. Preparing a Macro for Encryption
6.2. Creating an Encrypted Macro
6.3. Running an Encrypted Macro

6.1. Preparing a Macro for Encryption

Before encrypting a macro, you first create and debug the macro as usual. When you create an encrypted macro, you are responsible for keeping the original source file. You cannot recreate the source file from an encrypted macro. You then add an /ENCRYPT command as the first line and last of the macro. The /ENCRYPT command for the first line of the macro has the following syntax:

/ENCRYPT,Encryption_key,File_name,File_ext,Directory_Path/

Where

- **Encryption_key** is an eight-character password.
- **File_name** is the name of the encrypted macro filename.
- **File_ext** is an optional file extension for the encrypted macro file. If you want users to execute the macro as an "unknown" command, you should use the .mac extension.
- **Directory_Path** is the optional directory path that can contain up to 60 characters; you only need this argument if you do not want to write the encrypted macro file to your "home" directory.

Note the placement of the /ENCRYPT commands at the top and bottom of the listing in the following example:

```
/encrypt,mypasswd,myenfile,mac,macros/
/nopr
/encrypt
/prep7
/view,,1,-2,-3
/block,,arg1,,arg2,,arg3
/sphere,,arg4
/vsbv,1,2
/gopr
/encrypt
```

The /ENCRYPT command at the top of the macro instructs ANSYS to encrypt the file and use the string "mypasswd" as the encryption key. It will create an encrypted macro file called myenfile.mac and place it in the /macros subdirectory of the home directory. The /ENCRYPT command at the bottom instructs ANSYS to stop the encryption process and write the encrypted macro to the specified file.
The encrypted macro uses a `/NOPR` command as its second line to turn off echoing of ANSYS commands to the session log file. This is important if you wish to prevent users from reading the contents of the macro from the session log. It's a good practice to reactivate the session log by issuing the `/GOPR` command as the last command in the macro before the ending `/ENCRYPT` command.

### 6.2. Creating an Encrypted Macro

After putting the `/ENCRYPT` commands at the top and bottom of the macro, you can proceed to create the encrypted version of the macro. To do this, simply execute the macro through ANSYS. ANSYS will create the encrypted version with the name and location you specified through the `/ENCRYPT` command at the top of the macro. The result should look something like this

```
/DECRYPT, mypasswd

013^2, ^b
02x^0Se|Lv(yT.6>?  
03Jj3\_LuXd3=6=m+f$h]{eB
04:^V7S#c>
05daV;u(yY
06Tj3wJZ
/DECRYPT
```

Note that the individual commands within the macro are now encrypted, and that the encrypted material is enclosed by `/DECRYPT` commands. The encryption key is the argument to the first `/DECRYPT` command.

### 6.3. Running an Encrypted Macro

You can run an encrypted macro just as you would any other macro; place the encrypted macro within the macro search path. If you would prefer to run the encrypted macro without having the encryption key resident in the macro file, you can define the key as a "global encryption key" within ANSYS. To do this you must first replace the encryption key argument in the `/DECRYPT` command with the parameter PASSWORD. Thus, the first line of the encrypted macro becomes:

```
/DECRYPT, PASSWORD
```

Before executing the macro within ANSYS, issue the following command through the ANSYS Input command line:

```
/DECRYPT, PASSWORD, Encryption_Key
```

Where `Encryption_Key` is the encryption key used to encrypt the file. You can now execute the encrypted password. To delete the current global encryption key, issue the following ANSYS command:

```
/DECRYPT, PASSWORD, OFF
```
Chapter 7: APDL Commands

*ABBR
ABBRRES
ABBSAV
*AFUN
*ASK
*CFCLOSE
*CFOPEN
*CFWRITE
*CREATE
*CYCLE
*DEL
/DFLAB
*DIM
/DIRECTORY
*DO
*DOWHILE
*ELSE
*ELSEIF
*END
*ENDDO
*ENDIF
*EXIT
*GET
*GO
*IF
/INQUIRE
/MAIL
*MFOURII
*MFUN
/MKDIR
*MOPER
*MSG
*MWRITE
PARRES
PARSAV
/PMACRO
/PSEARCH
*REPEAT
*RETURN
/RMDIR
*SET
*SREAD
*STATUS
*TAXIS
/TEE
*TOPER
*TREAD
/UCMD
*ULIB
*USE
*VABS
*VCOL
*VCUM
*VEDIT
*VFACT
*VFILL
*VFUN
*VGET
*VITRP
*VLEN
*VMASK
*VOPER
*VPLOT
*VPUT
*VREAD
*VSCFUN
*VSTAT
*VWRITE
/WAIT
Appendix A. APDL Gateway Commands

When you need to determine the applicability of a command or a group of commands to a specific product, the following *GET functions will return a TRUE or a FALSE (a 1 or a 0) value to indicate if the command in question is valid for your ANSYS product.

Additional *get commands for a new entity=PRODUCT

<table>
<thead>
<tr>
<th>Item1</th>
<th>It1num</th>
<th>Item2</th>
<th>It2num</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pname</td>
<td></td>
<td>name</td>
<td></td>
<td>-P option from Ansys command line</td>
</tr>
<tr>
<td></td>
<td></td>
<td>start</td>
<td>1-n</td>
<td>Ansys product name. A character string of 8 characters is returned starting at position It2num. Use *dim and *do to get all 32 characters.</td>
</tr>
</tbody>
</table>

Entity=PRODUCT,ENTNUM=0 (or blank)

<table>
<thead>
<tr>
<th>Item1</th>
<th>It1num</th>
<th>Description (return values: 1=allowed, 0=not allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>/aux12</td>
<td></td>
<td>Check for Ansys gateway command/feature /AUX12</td>
</tr>
<tr>
<td>/config</td>
<td></td>
<td>Check for Ansys gateway command/feature /CONFIG</td>
</tr>
<tr>
<td>/ucmd</td>
<td></td>
<td>Check for Ansys gateway command/feature /UCMD</td>
</tr>
<tr>
<td>addam</td>
<td></td>
<td>Check for Ansys gateway command/feature ADDAM</td>
</tr>
<tr>
<td>alphad</td>
<td></td>
<td>Check for Ansys gateway command/feature ALPHAD</td>
</tr>
<tr>
<td>antype</td>
<td></td>
<td>Check for Ansys gateway command/feature ANTYPE</td>
</tr>
<tr>
<td>antype static</td>
<td></td>
<td>Check for Ansys gateway command/feature ANTYPE,STATIC</td>
</tr>
<tr>
<td>antype buckle</td>
<td></td>
<td>Check for Ansys gateway command/feature ANTYPE,BUCKLE</td>
</tr>
<tr>
<td>antype modal</td>
<td></td>
<td>Check for Ansys gateway command/feature ANTYPE,MODAL</td>
</tr>
<tr>
<td>antype harmic</td>
<td></td>
<td>Check for Ansys gateway command/feature ANTYPE,HARMIC</td>
</tr>
<tr>
<td>antype trans</td>
<td></td>
<td>Check for Ansys gateway command/feature ANTYPE,TRANS</td>
</tr>
<tr>
<td>antype subtr</td>
<td></td>
<td>Check for Ansys gateway command/feature ANTYPE,SUBSTR</td>
</tr>
<tr>
<td>antype spectr</td>
<td></td>
<td>Check for Ansys gateway command/feature ANTYPE,SPECTR</td>
</tr>
<tr>
<td>arclen</td>
<td></td>
<td>Check for Ansys gateway command/feature ARCLEN</td>
</tr>
<tr>
<td>betad</td>
<td></td>
<td>Check for Ansys gateway command/feature BETAD</td>
</tr>
<tr>
<td>blc4</td>
<td></td>
<td>Check for Ansys gateway command/feature BLC4</td>
</tr>
<tr>
<td>blc5</td>
<td></td>
<td>Check for Ansys gateway command/feature BLC5</td>
</tr>
<tr>
<td>block</td>
<td></td>
<td>Check for Ansys gateway command/feature BLOCK</td>
</tr>
<tr>
<td>cdread</td>
<td></td>
<td>Check for Ansys gateway command/feature CDREAD</td>
</tr>
<tr>
<td>con4</td>
<td></td>
<td>Check for Ansys gateway command/feature CON4</td>
</tr>
<tr>
<td>cone</td>
<td></td>
<td>Check for Ansys gateway command/feature CONE</td>
</tr>
<tr>
<td>cqc</td>
<td></td>
<td>Check for Ansys gateway command/feature CQC</td>
</tr>
<tr>
<td>cyl4</td>
<td></td>
<td>Check for Ansys gateway command/feature CYL4</td>
</tr>
<tr>
<td>cyl5</td>
<td></td>
<td>Check for Ansys gateway command/feature CYL5</td>
</tr>
<tr>
<td>cylind</td>
<td></td>
<td>Check for Ansys gateway command/feature CYLIND</td>
</tr>
</tbody>
</table>

Entity=PRODUCT,ENTNUM=0 (or blank)

<table>
<thead>
<tr>
<th>Item1</th>
<th>It1num</th>
<th>Description (return values: 1=allowed, 0=not allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>damorph</td>
<td></td>
<td>Check for Ansys gateway command/feature DAMORPH</td>
</tr>
<tr>
<td>demorph</td>
<td></td>
<td>Check for Ansys gateway command/feature DEMORPH</td>
</tr>
<tr>
<td>dsum</td>
<td></td>
<td>Check for Ansys gateway command/feature DSUM</td>
</tr>
<tr>
<td>dvmorph</td>
<td></td>
<td>Check for Ansys gateway command/feature DVMORPH</td>
</tr>
<tr>
<td>edadapt</td>
<td></td>
<td>Check for Ansys gateway command/feature EADAPT</td>
</tr>
<tr>
<td>edbvis</td>
<td></td>
<td>Check for Ansys gateway commands/feature EDBVIS</td>
</tr>
<tr>
<td>eddc</td>
<td></td>
<td>Check for Ansys gateway commands/feature EDDC</td>
</tr>
<tr>
<td>edcgen</td>
<td></td>
<td>Check for Ansys gateway commands/feature EDCGEN</td>
</tr>
<tr>
<td>edclist</td>
<td></td>
<td>Check for Ansys gateway commands/feature EDCLIST</td>
</tr>
<tr>
<td>edcontact</td>
<td></td>
<td>Check for Ansys gateway commands/feature EDCONTACT</td>
</tr>
<tr>
<td>edcpu</td>
<td></td>
<td>Check for Ansys gateway commands/feature EDCPU</td>
</tr>
<tr>
<td>edcrb</td>
<td></td>
<td>Check for Ansys gateway commands/feature EDCRB</td>
</tr>
<tr>
<td>edcsc</td>
<td></td>
<td>Check for Ansys gateway commands/feature EDCSC</td>
</tr>
<tr>
<td>edcts</td>
<td></td>
<td>Check for Ansys gateway commands/feature EDCTS</td>
</tr>
</tbody>
</table>
### Appendix A. APDL Gateway Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Gateway Commands/Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>edcurve</td>
<td>Check for Ansys gateway commands/feature EDCURVE</td>
<td></td>
</tr>
<tr>
<td>eddamp</td>
<td>Check for Ansys gateway commands/feature EDDAMP</td>
<td></td>
</tr>
<tr>
<td>edenergy</td>
<td>Check for Ansys gateway commands/feature EDENERGY</td>
<td></td>
</tr>
<tr>
<td>edfplot</td>
<td>Check for Ansys gateway commands/feature EDFPLOT</td>
<td></td>
</tr>
<tr>
<td>edhgl</td>
<td>Check for Ansys gateway commands/feature EDHGLS</td>
<td></td>
</tr>
<tr>
<td>edhtime</td>
<td>Check for Ansys gateway commands/feature EDHTIME</td>
<td></td>
</tr>
<tr>
<td>edint</td>
<td>Check for Ansys gateway commands/feature EDINT</td>
<td></td>
</tr>
<tr>
<td>edvel</td>
<td>Check for Ansys gateway commands/feature EDVEL</td>
<td></td>
</tr>
<tr>
<td>edlcs</td>
<td>Check for Ansys gateway commands/feature EDLCS</td>
<td></td>
</tr>
<tr>
<td>edldplot</td>
<td>Check for Ansys gateway commands/feature EDLDPLOT</td>
<td></td>
</tr>
</tbody>
</table>

**Entity=PRODUCT,ENTNUM=0 (or blank)**

<table>
<thead>
<tr>
<th>Item</th>
<th>It1num</th>
<th>Description</th>
<th>Return Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>edload</td>
<td></td>
<td>Check for Ansys gateway commands/feature EDLOAD</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>edmp</td>
<td></td>
<td>Check for Ansys gateway commands/feature EDMMP</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>ednb</td>
<td></td>
<td>Check for Ansys gateway commands/feature EDNBD</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>edntad</td>
<td></td>
<td>Check for Ansys gateway commands/feature EDNNTAD</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>edout</td>
<td></td>
<td>Check for Ansys gateway commands/feature EDOUT</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>edpart</td>
<td></td>
<td>Check for Ansys gateway commands/feature EDPART</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>edread</td>
<td></td>
<td>Check for Ansys gateway commands/feature EDDRREAD</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>eddrelax</td>
<td></td>
<td>Check for Ansys gateway commands/feature EDDRELLAX</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>edrst</td>
<td></td>
<td>Check for Ansys gateway commands/feature EDRST</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>edshell</td>
<td></td>
<td>Check for Ansys gateway commands/feature ED SHELL</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>edsolve</td>
<td></td>
<td>Check for Ansys gateway commands/feature EDSOLVE</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>edstart</td>
<td></td>
<td>Check for Ansys gateway commands/feature EDSSTART</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>edweld</td>
<td></td>
<td>Check for Ansys gateway commands/feature EDWELD</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>edwrite</td>
<td></td>
<td>Check for Ansys gateway commands/feature EDWRITE</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>emis</td>
<td></td>
<td>Check for Ansys gateway commands/feature EMIS</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>et</td>
<td></td>
<td>Check for Ansys gateway commands/feature ET</td>
<td>1=allowed, 0=not allowed</td>
</tr>
</tbody>
</table>

**Entity=PRODUCT,ENTNUM=0 (or blank)**

<table>
<thead>
<tr>
<th>Item</th>
<th>It1num</th>
<th>Description</th>
<th>Return Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>etchg</td>
<td></td>
<td>Check for Ansys gateway commands/feature ETCCHG</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>fldata</td>
<td></td>
<td>Check for Ansys gateway commands/feature FLDATA</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>flotest</td>
<td></td>
<td>Check for Ansys gateway commands/feature FLOTTEST</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>flread</td>
<td></td>
<td>Check for Ansys gateway commands/feature FLREAD</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>fvmesh</td>
<td></td>
<td>Check for Ansys gateway commands/feature FVMESH</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>grp</td>
<td></td>
<td>Check for Ansys gateway commands/feature GRP</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>hropt</td>
<td></td>
<td>Check for Ansys gateway commands/feature HROPT</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>hropt</td>
<td></td>
<td>Check for Ansys gateway commands/feature HROPT,FULL</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>hropt</td>
<td></td>
<td>Check for Ansys gateway commands/feature HROPT,REDUC</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>hropt</td>
<td></td>
<td>Check for Ansys gateway commands/feature HROPT,MSUP</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>igesin</td>
<td></td>
<td>Check for Ansys gateway commands/feature IGESIN</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>igesout</td>
<td></td>
<td>Check for Ansys gateway commands/feature IGESOUT</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>modopt</td>
<td></td>
<td>Check for Ansys gateway commands/feature MODOPT</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>modopt</td>
<td></td>
<td>Check for Ansys gateway commands/feature MODOPT,REDUC</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>modopt</td>
<td></td>
<td>Check for Ansys gateway commands/feature MODOPT,SUBSP</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>modopt</td>
<td></td>
<td>Check for Ansys gateway commands/feature MODOPT,UNS YM</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>modopt</td>
<td></td>
<td>Check for Ansys gateway commands/feature MODOPT,DAMP</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>modopt</td>
<td></td>
<td>Check for Ansys gateway commands/feature MODOPT,LANB</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>modopt</td>
<td></td>
<td>Check for Ansys gateway commands/feature MODOPT,QRDAMP</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>modopt</td>
<td></td>
<td>Check for Ansys gateway commands/feature MODOPT MOONEY</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>mp</td>
<td></td>
<td>Check for Ansys gateway commands/feature MP</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>mp</td>
<td></td>
<td>Check for Ansys gateway commands/feature MP,EX</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>mp</td>
<td></td>
<td>Check for Ansys gateway commands/feature MP,ALPX</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>mp</td>
<td></td>
<td>Check for Ansys gateway commands/feature MP,REFT</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>mp</td>
<td></td>
<td>Check for Ansys gateway commands/feature MP,PRXY</td>
<td>1=allowed, 0=not allowed</td>
</tr>
</tbody>
</table>

**Entity=PRODUCT,ENTNUM=0 (or blank)**

<table>
<thead>
<tr>
<th>Item</th>
<th>It1num</th>
<th>Description</th>
<th>Return Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>mp</td>
<td></td>
<td>Check for Ansys gateway commands/feature MP</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>mp</td>
<td></td>
<td>Check for Ansys gateway commands/feature MP,NUXY</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>mp</td>
<td></td>
<td>Check for Ansys gateway commands/feature MP,GXY</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>mp</td>
<td></td>
<td>Check for Ansys gateway commands/feature MP,DAMP</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>mp</td>
<td></td>
<td>Check for Ansys gateway commands/feature MP,MU</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>mp</td>
<td></td>
<td>Check for Ansys gateway commands/feature MP,DENS</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>mp</td>
<td></td>
<td>Check for Ansys gateway commands/feature MP,C</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>mp</td>
<td></td>
<td>Check for Ansys gateway commands/feature MP,ENTH</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>mp</td>
<td></td>
<td>Check for Ansys gateway commands/feature MP,KXX</td>
<td>1=allowed, 0=not allowed</td>
</tr>
<tr>
<td>Item1</td>
<td>It1num</td>
<td>Description (return values: 1=allowed, 0=not allowed)</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
<td>-----------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>msquad</td>
<td></td>
<td>Check for Ansys gateway commands/feature MSQUAD</td>
<td></td>
</tr>
<tr>
<td>mrsrelax</td>
<td></td>
<td>Check for Ansys gateway commands/feature MRSRELAX</td>
<td></td>
</tr>
<tr>
<td>mssolu</td>
<td></td>
<td>Check for Ansys gateway commands/feature MSSOLO</td>
<td></td>
</tr>
<tr>
<td>msspec</td>
<td></td>
<td>Check for Ansys gateway commands/feature MSSPEC</td>
<td></td>
</tr>
<tr>
<td>msvary</td>
<td></td>
<td>Check for Ansys gateway commands/feature MSVARY</td>
<td></td>
</tr>
<tr>
<td>nlgeom</td>
<td></td>
<td>Check for Ansys gateway commands/feature NLGEOM</td>
<td></td>
</tr>
<tr>
<td>nrlsum</td>
<td></td>
<td>Check for Ansys gateway commands/feature NRLSUM</td>
<td></td>
</tr>
<tr>
<td>optyp</td>
<td></td>
<td>Check for Ansys gateway commands/feature OPTYP</td>
<td></td>
</tr>
<tr>
<td>optyp</td>
<td></td>
<td>Check for Ansys gateway commands/feature OPTYP,SUBP</td>
<td></td>
</tr>
<tr>
<td>optyp</td>
<td></td>
<td>Check for Ansys gateway commands/feature OPTYP,FIRST</td>
<td></td>
</tr>
<tr>
<td>optyp</td>
<td></td>
<td>Check for Ansys gateway commands/feature OPTYP,RAND</td>
<td></td>
</tr>
<tr>
<td>optyp</td>
<td></td>
<td>Check for Ansys gateway commands/feature OPTYP,FECT</td>
<td></td>
</tr>
<tr>
<td>optyp</td>
<td></td>
<td>Check for Ansys gateway commands/feature OPTYP,GRAD</td>
<td></td>
</tr>
<tr>
<td>optyp</td>
<td></td>
<td>Check for Ansys gateway commands/feature OPTYP,SLEEP</td>
<td></td>
</tr>
<tr>
<td>opuser</td>
<td></td>
<td>Check for Ansys gateway commands/feature OPUSER</td>
<td></td>
</tr>
<tr>
<td>pri2</td>
<td></td>
<td>Check for Ansys gateway commands/feature PRI2</td>
<td></td>
</tr>
<tr>
<td>prism</td>
<td></td>
<td>Check for Ansys gateway commands/feature PRISM</td>
<td></td>
</tr>
<tr>
<td>psdcom</td>
<td></td>
<td>Check for Ansys gateway commands/feature PSDCOM</td>
<td></td>
</tr>
<tr>
<td>psdfreq</td>
<td></td>
<td>Check for Ansys gateway commands/feature PSDFREQ</td>
<td></td>
</tr>
<tr>
<td>psolve</td>
<td></td>
<td>Check for Ansys gateway commands/feature PSOLVE</td>
<td></td>
</tr>
<tr>
<td>rate</td>
<td></td>
<td>Check for Ansys gateway commands/feature RATE</td>
<td></td>
</tr>
<tr>
<td>resume</td>
<td></td>
<td>Check for Ansys gateway commands/feature RESUME</td>
<td></td>
</tr>
</tbody>
</table>
### Appendix A. APDL Gateway Commands

<table>
<thead>
<tr>
<th>Item</th>
<th>It1num</th>
<th>Description (return values: 1=allowed, 0=not allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>rpr4</td>
<td></td>
<td>Check for Ansys gateway commands/feature RPR4</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Entity=PRODUCT,ENTNUM=0 (or blank)</strong></td>
</tr>
</tbody>
</table>

| rprism |        | Check for Ansys gateway commands/feature RPRISM        |
| save   |        | Check for Ansys gateway commands/feature SAVE          |
| se     |        | Check for Ansys gateway commands/feature SE            |
| sesymm |        | Check for Ansys gateway commands/feature SESYN         |
| setran |        | Check for Ansys gateway commands/feature SETRAN        |
| solve  |        | Check for Ansys gateway commands/feature SOLVE         |
| sph4   |        | Check for Ansys gateway commands/feature SPH4          |
| sph5   |        | Check for Ansys gateway commands/feature SPH5          |
| sphere |        | Check for Ansys gateway commands/feature SPHERE        |
| spop   |        | Check for Ansys gateway commands/feature SPPO          |
| spop   | sprs   | Check for Ansys gateway commands/feature SPOP,SPRS     |
| spop   | mprs   | Check for Ansys gateway commands/feature SPOP,MPRS     |
| spop   | ddam   | Check for Ansys gateway commands/feature SPOP,DDAM     |
| spop   | psd    | Check for Ansys gateway commands/feature SPOP,PSD      |
| sesym  |        | Check for Ansys gateway commands/feature SESYMN        |
| setran |        | Check for Ansys gateway commands/feature SETRAN        |
| solve  |        | Check for Ansys gateway commands/feature SOLVE         |
| sph4   |        | Check for Ansys gateway commands/feature SPH4          |
| sph5   |        | Check for Ansys gateway commands/feature SPH5          |
| sphere |        | Check for Ansys gateway commands/feature SPHERE        |
| spop   |        | Check for Ansys gateway commands/feature SPPO          |
| spop   | sprs   | Check for Ansys gateway commands/feature SPOP,SPRS     |
| spop   | mprs   | Check for Ansys gateway commands/feature SPOP,MPRS     |
| spop   | ddam   | Check for Ansys gateway commands/feature SPOP,DDAM     |
| spop   | psd    | Check for Ansys gateway commands/feature SPOP,PSD      |
| sesym  |        | Check for Ansys gateway commands/feature SESYMN        |
| setran |        | Check for Ansys gateway commands/feature SETRAN        |
| solve  |        | Check for Ansys gateway commands/feature SOLVE         |
| sph4   |        | Check for Ansys gateway commands/feature SPH4          |
| sph5   |        | Check for Ansys gateway commands/feature SPH5          |
| sphere |        | Check for Ansys gateway commands/feature SPHERE        |
| spop   |        | Check for Ansys gateway commands/feature SPPO          |
| spop   | sprs   | Check for Ansys gateway commands/feature SPOP,SPRS     |
| spop   | mprs   | Check for Ansys gateway commands/feature SPOP,MPRS     |
| spop   | ddam   | Check for Ansys gateway commands/feature SPOP,DDAM     |
| spop   | psd    | Check for Ansys gateway commands/feature SPOP,PSD      |
| sesym  |        | Check for Ansys gateway commands/feature SESYMN        |
| setran |        | Check for Ansys gateway commands/feature SETRAN        |
| solve  |        | Check for Ansys gateway commands/feature SOLVE         |

**Release 12.0 © 2009 SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.**
Appendix A. APDL Gateway Commands

vmesh    Check for Ansys gateway commands/feature VMESH
voffset  Check for Ansys gateway commands/feature VOFFSET
vovlap   Check for Ansys gateway commands/feature VOVLAP
vptn     Check for Ansys gateway commands/feature VPTN
vrotat   Check for Ansys gateway commands/feature VROTAT
vsba     Check for Ansys gateway commands/feature VSBA
vsbv     Check for Ansys gateway commands/feature VSBV
vsbw     Check for Ansys gateway commands/feature VSBW
vsymm    Check for Ansys gateway commands/feature VSYMM
vtran    Check for Ansys gateway commands/feature VTRAN

Entity=PRODUCT,ENTNUM=0 (or blank)
Item1    It1num      Description (return values:  1=allowed,  0=not allowed)
elem     i           Check to see if Ansys element type "i" is allowed.

Entity=PRODUCT,ENTNUM=0 (or blank)
Item1    It1num      Description (return values:  1=allowed,  0=not allowed)
limit    node       Get maximum allowed node number
limit    elem       Get maximum allowed element number
limit    kp         Get maximum allowed keypoint number
limit    line       Get maximum allowed line number
limit    area       Get maximum allowed area number
limit    vol        Get maximum allowed volume number
limit    dof        Get maximum allowed dof number
limit    mdof       Get maximum allowed master dof number
Appendix B. GET Function Summary

A "get function" is available for some items, and can be used instead of the *GET command. The function returns the value and uses it where the function is input (bypassing the need for storing the value with a parameter name and inputting the parameter name where the value is to be used). For example, assume the average X-location of two nodes is to be calculated. Using the *GET command, parameter L1 can be assigned the X location of node 1 (*GET, L1, NODE, 1, LOC, X), and parameter L2 can be assigned the X location of node 2, then the mid location can be computed from MID = (L1+L2)/2. However, using the node location "get function" NX(N), which returns the X location of node N, MID can be computed directly from MID = (NX(1)+NX(2))/2, without the need for intermediate parameters L1 and L2. Get functions return values in the active coordinate system unless stated otherwise.

Get function arguments may themselves be parameters or other get functions. The get function NELEM(ENUM,NPOS) returns the node number in position NPOS for element ENUM. Combining functions, NX(NELEM(ENUM,NPOS)) returns the X location of that node. Get functions (where available) are shown with the corresponding *GET items in the tables below and are summarized at the end of this command description.

Get functions are described at the beginning of this command (see the section titled Notes) and are shown as alternatives to the *GET items where they apply. They are summarized here (grouped by functionality) for convenience.

Table B.1 *GET - Get Function Summary

<table>
<thead>
<tr>
<th>Entity Status Get Function</th>
<th>&quot;Get Function&quot; Summary</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSEL(N)</td>
<td>Status of node N: -1=unselected, 0=undefined, 1=selected.</td>
<td></td>
</tr>
<tr>
<td>ESEL(E)</td>
<td>Status of element E: -1=unselected, 0=undefined, 1=selected.</td>
<td></td>
</tr>
<tr>
<td>KSEL(K)</td>
<td>Status of keypoint K: -1=unselected, 0=undefined, 1=selected.</td>
<td></td>
</tr>
<tr>
<td>LSEL(L)</td>
<td>Status of line L: -1=unselected, 0=undefined, 1=selected.</td>
<td></td>
</tr>
<tr>
<td>ASEL(A)</td>
<td>Status of area A: -1=unselected, 0=undefined, 1=selected.</td>
<td></td>
</tr>
<tr>
<td>VSEL(V)</td>
<td>Status of volume V: -1=unselected, 0=undefined, 1=selected.</td>
<td></td>
</tr>
</tbody>
</table>

**Next Selected Entity**

<table>
<thead>
<tr>
<th>Next Selected Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDNEXT(N)</td>
<td>Next selected node having a node number greater than N.</td>
</tr>
<tr>
<td>ELNEXT(E)</td>
<td>Next selected element having an element number greater than E.</td>
</tr>
<tr>
<td>KPNEXT(K)</td>
<td>Next selected keypoint having a keypoint number greater than K.</td>
</tr>
<tr>
<td>LSNEXT(L)</td>
<td>Next selected line having a line number greater than L.</td>
</tr>
<tr>
<td>ARNEXT(A)</td>
<td>Next selected area having an area number greater than A.</td>
</tr>
</tbody>
</table>
### "Get Function" Summary

<table>
<thead>
<tr>
<th>Entity Status Get Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>VLNEXT(V)</strong></td>
<td>Next selected volume having a volume number greater than ( V ).</td>
</tr>
<tr>
<td><strong>Locations</strong></td>
<td></td>
</tr>
<tr>
<td><strong>CENTRX(E)</strong></td>
<td>Centroid X-coordinate of element ( E ) in global Cartesian coordinate system. Centroid is determined from the selected nodes on the element.</td>
</tr>
<tr>
<td><strong>CENTRY(E)</strong></td>
<td>Centroid Y-coordinate of element ( E ) in global Cartesian coordinate system. Centroid is determined from the selected nodes on the element.</td>
</tr>
<tr>
<td><strong>CENTRZ(E)</strong></td>
<td>Centroid Z-coordinate of element ( E ) in global Cartesian coordinate system. Centroid is determined from the selected nodes on the element.</td>
</tr>
<tr>
<td><strong>NX(N)</strong></td>
<td>X-coordinate of node ( N ) in the active coordinate system.</td>
</tr>
<tr>
<td><strong>NY(N)</strong></td>
<td>Y-coordinate of node ( N ) in the active coordinate system.</td>
</tr>
<tr>
<td><strong>NZ(N)</strong></td>
<td>Z-coordinate of node ( N ) in the active coordinate system.</td>
</tr>
<tr>
<td><strong>KX(K)</strong></td>
<td>X-coordinate of keypoint ( K ) in the active coordinate system</td>
</tr>
<tr>
<td><strong>KY(K)</strong></td>
<td>Y-coordinate of keypoint ( K ) in the active coordinate system</td>
</tr>
<tr>
<td><strong>KZ(K)</strong></td>
<td>Z-coordinate of keypoint ( K ) in the active coordinate system</td>
</tr>
<tr>
<td><strong>LX(L, LFRAC)</strong></td>
<td>X-coordinate of line ( L ) at length fraction ( LFRAC ) (0.0 to 1.0).</td>
</tr>
<tr>
<td><strong>LY(L, LFRAC)</strong></td>
<td>Y-coordinate of line ( L ) at length fraction ( LFRAC ) (0.0 to 1.0).</td>
</tr>
<tr>
<td><strong>LZ(L, LFRAC)</strong></td>
<td>Z-coordinate of line ( L ) at length fraction ( LFRAC ) (0.0 to 1.0).</td>
</tr>
<tr>
<td><strong>LSX(L, LFRAC)</strong></td>
<td>X slope of line ( L ) at length fraction ( LFRAC ) (0.0 to 1.0).</td>
</tr>
<tr>
<td><strong>LSY(L, LFRAC)</strong></td>
<td>Y slope of line ( L ) at length fraction ( LFRAC ) (0.0 to 1.0).</td>
</tr>
<tr>
<td><strong>LSZ(L, LFRAC)</strong></td>
<td>Z slope of line ( L ) at length fraction ( LFRAC ) (0.0 to 1.0).</td>
</tr>
<tr>
<td><strong>Nearest to Location</strong></td>
<td></td>
</tr>
<tr>
<td><strong>NODE(X, Y, Z)</strong></td>
<td>Number of the selected node nearest the ( X, Y, Z ) point (in the active coordinate system, lowest number for coincident nodes).</td>
</tr>
<tr>
<td><strong>KP(X, Y, Z)</strong></td>
<td>Number of the selected keypoint nearest the ( X, Y, Z ) point (in the active coordinate system, lowest number for coincident nodes).</td>
</tr>
<tr>
<td><strong>Distances</strong></td>
<td></td>
</tr>
<tr>
<td><strong>DISTND(N1, N2)</strong></td>
<td>Distance between nodes ( N1 ) and ( N2 ).</td>
</tr>
<tr>
<td><strong>DISTKP(K1, K2)</strong></td>
<td>Distance between keypoints ( K1 ) and ( K2 ).</td>
</tr>
<tr>
<td><strong>DISTEN(E, N)</strong></td>
<td>Distance between the centroid of element ( E ) and node ( N ). Centroid is determined from the selected nodes on the element.</td>
</tr>
<tr>
<td>*<em>Angles (in radians by default -- see the <em>AFUN command)</em></em></td>
<td></td>
</tr>
<tr>
<td><strong>ANGLEN(N1, N2, N3)</strong></td>
<td>Subtended angle between two lines (defined by three nodes where ( N1 ) is the vertex node). Default is in radians.</td>
</tr>
</tbody>
</table>
### Get Function Summary

<table>
<thead>
<tr>
<th>Entity Status Get Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANGLEK(K1, K2, K3)</td>
<td>Subtended angle between two lines (defined by three keypoints where K1 is the vertex keypoint). Default is in radians.</td>
</tr>
<tr>
<td>Nearest to Entity</td>
<td></td>
</tr>
<tr>
<td>NNEAR(N)</td>
<td>Selected node nearest node N.</td>
</tr>
<tr>
<td>KNEAR(K)</td>
<td>Selected keypoint nearest keypoint K.</td>
</tr>
<tr>
<td>ENEAR(N)</td>
<td>Selected element nearest node N. The element position is calculated from the selected nodes.</td>
</tr>
<tr>
<td>Areas</td>
<td></td>
</tr>
<tr>
<td>AREAND(N1, N2, N3)</td>
<td>Area of the triangle with vertices at nodes N1, N2, and N3.</td>
</tr>
<tr>
<td>AREAKP(K1, K2, K3)</td>
<td>Area of the triangle with vertices at keypoints K1, K2, and K3.</td>
</tr>
<tr>
<td>ARNODE(N)</td>
<td>Area at node N apportioned from selected elements attached to node N. For 2-D planar solids, returns edge area associated with the node. For axisymmetric solids, returns edge surface area associated with the node. For 3-D volumetric solids, returns face area associated with the node. For 3-D, select all the nodes of the surface of interest before using ARNODE.</td>
</tr>
<tr>
<td>Normals</td>
<td></td>
</tr>
<tr>
<td>NORMNX(N1, N2, N3)</td>
<td>X-direction cosine of the normal to the plane containing nodes N1, N2, and N3.</td>
</tr>
<tr>
<td>NORMNY(N1, N2, N3)</td>
<td>Y-direction cosine of the normal to the plane containing nodes N1, N2, and N3.</td>
</tr>
<tr>
<td>NORMNZ(N1, N2, N3)</td>
<td>Z-direction cosine of the normal to the plane containing nodes N1, N2, and N3.</td>
</tr>
<tr>
<td>NORMKX(K1, K2, K3)</td>
<td>X-direction cosine of the normal to the plane containing keypoints K1, K2, and K3.</td>
</tr>
<tr>
<td>NORMKY(K1, K2, K3)</td>
<td>Y-direction cosine of the normal to the plane containing keypoints K1, K2, and K3.</td>
</tr>
<tr>
<td>NORMKZ(K1, K2, K3)</td>
<td>Z-direction cosine of the normal to the plane containing keypoints K1, K2, and K3.</td>
</tr>
<tr>
<td>Connectivity</td>
<td></td>
</tr>
<tr>
<td>ENEXTN(N, LOC)</td>
<td>Element connected to node N. LOC is the position in the resulting list when many elements share the node. A zero is returned at the end of the list.</td>
</tr>
<tr>
<td>NELEM(E, NPOS)</td>
<td>Node number in position NPOS (1--20) of element E.</td>
</tr>
<tr>
<td>NODEDOF(N)</td>
<td>Returns the bit pattern for the active DOFs at the specified node.</td>
</tr>
</tbody>
</table>

- bit 0 is UX, bit 1 is UY,... bit 5 is ROTZ
- bits 6,7,8 are AX,AY,AZ
- bits 9,10,11 are VX,VY,VZ
- bit 18 is PRES, bit 19 is TEMP, bit 20 is VOLT, bit 21 is MAG
- bit 24 is EMF, bit 25 is CURR
"Get Function" Summary

<table>
<thead>
<tr>
<th>Entity Status Get Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>For a node with UX,UY,UZ the return value will be 7 (bits 0,1,2) For a node with UX,UY,UZ,ROTX,ROTY,ROTZ the return value will be 63 (bits 0,1,2,3,4,5)</td>
</tr>
</tbody>
</table>

**Faces**

**ELADJ(E, FACE)**

For 2-D planar solids and 3-D volumetric solids, element adjacent to a face (FACE) of element E. The face number is the same as the surface load key number. Only elements of the same dimensionality and shape are considered. A -1 is returned if more than one is adjacent.

**NDFACE(E, FACE, LOC)**

Node in position LOC of a face number FACE of element E. The face number is the same as the surface load key number. LOC is the nodal position on the face (for an UJK face, LOC=1 is at node I, 2 is at node J, etc.)

**NMFACE(E)**

Face number of element E containing the selected nodes. The face number output is the surface load key. If multiple load keys occur on a face (such as for line and area elements) the lowest load key for that face is output.

**ARFACE(E)**

For 2-D planar solids and 3-D volumetric solids, returns the area of the face of element E containing the selected nodes. For axisymmetric elements, the area is the full (360 degree) area.

**Degree of Freedom Results**

**UX(N)**

UX structural displacement at node N.

**UY(N)**

UY structural displacement at node N.

**UZ(N)**

UZ structural displacement at node N.

**ROTX (N)**

ROTX structural rotation at node N.

**ROTY (N)**

ROTY structural rotation at node N.

**ROTZ (N)**

ROTZ structural rotation at node N.

**TEMP(N)**

Temperature at node N. For SHELL131 and SHELL132 elements with KEYOPT(3) = 0 or 1, use TBOT(N), TE2(N), TE3(N), . . ., TTOP(N) instead of TEMP(N).

**PRES(N)**

Pressure at node N.

**VX(N)**

VX fluid velocity at node N.

**VY(N)**

VY fluid velocity at node N.

**VZ (N)**

VZ fluid velocity at node N.

**ENKE(N)**

Turbulent kinetic energy (FLOTRAN) at node N.

**ENDS(N)**

Turbulent energy dissipation (FLOTRAN) at node N.

**VOLT(N)**

Electric potential at node N.

**MAG(N)**

Magnetic scalar potential at node N.

**AX(N)**

AX magnetic vector potential at node N.

**AY(N)**

AY magnetic vector potential at node N.
**"Get Function" Summary**

<table>
<thead>
<tr>
<th>Entity Status Get Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ((N))</td>
<td>AZ magnetic vector potential at node (N).</td>
</tr>
</tbody>
</table>

**Returns information about the data base manager**

- VIRTINQR(1) | Number of pages in core. |
- VIRTINQR(4) | Page size in integer words. |
- VIRTINQR(7) | Maximum number of pages allowed on disk. |
- VIRTINQR(8) | Number of read/write operations on page. |
- VIRTINQR(9) | Maximum record number on page. |
- VIRTINQR(11) | Maximum pages touched. |

**Returns the current value of ANSYS filtering keywords.**

KWGET(\(KEYWORD\)) | Returns the current value the keyword specified by \(KEYWORD\). See the ANSYS UIDL Programmer's Guide for a list of keywords and values. |

**Character String Functions** Strings must be dimensioned (see *DIM) as a character parameter or enclosed in single apostrophes ('char').

**Functions which return a double precision value of a numeric character string.**

- VALCHR(\(a8\)) \(a8\) is a decimal value expressed in a string. |
- VALOCT(\(a8\)) \(a8\) is an octal value expressed in a string. |
- VALHEX(\(a8\)) \(a8\) is a hex value expressed in a string. |

**Functions which return an 8 character string of a numeric value.**

- CHRVAL(\(dp\)) \(dp\) is a double precision variable. |
- CHROCT(\(dp\)) \(dp\) is an integer value. |
- CHRHEX(\(dp\)) \(dp\) is an integer value. |

**Functions which manipulate strings:** \(StrOut\) is the output string (or character parameter) \(Str1\) and \(Str2\) are input strings. Strings are a maximum of 128 characters. (see *DIM)

- StrOut = STRSUB(\(Str1, nLoc, nChar\)) Get the nChar substring starting at character nLoc in Str1. |
- StrOut = STRCAT(\(Str1, Str2\)) Add Str2 at the end of Str1. |
- StrOut = STRFILL(\(Str1, Str2, nLoc\)) Add Str2 to Str1 starting at character nLoc. |
- StrOut = STRCOMP(\(Str1\)) Remove all blanks from Str1 |
- nLoc = STRLEFT(\(Str1\)) Left-justify Str1 |
- nLoc = STRLENG(\(Str1, Str2\)) Get starting location of Str2 in Str1. |
- StrOut = UPCASE(\(Str1\)) Upper case of Str1 |
- StrOut = LWCASE(\(Str1\)) Lower case of Str1 |

**The following functions manipulate file names.**

- Path String = JOIN ('directory','filename','extension') Produces a contiguous pathstring. e.g. directory/filename.ext |
- Path String = JOIN ('directory','filename') Produces a contiguous pathstring. e.g. directory/filename |
- SPLIT('PathString', 'DIR') Produces a separate output of the directory from the pathstring.
**"Get Function" Summary**

<table>
<thead>
<tr>
<th>Entity Status Get Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPLIT('PathString', 'FILE')</td>
<td>Produces a separate output of the complete filename (with extension) from the pathstring.</td>
</tr>
<tr>
<td>SPLIT('PathString', 'NAME')</td>
<td>Produces a separate output of the filename from the pathstring.</td>
</tr>
<tr>
<td>SPLIT('PathString', 'EXT')</td>
<td>Produces a separate output of the file extension from the pathstring.</td>
</tr>
</tbody>
</table>
Index

A
*ABBR command, 3, 15
*ABBRES command, 5
abbreviations
defined, 3
files, 5
nesting on toolbar, 5
*ABBSAV command, 5
*ABCHECK command
defined, 67
*ABFINI command, 67
*AFUN command, 16
ANSYS startup options, 9
ANSYS_MACROLIB environment variable, 47
APDL
comments, 15
defined, 1
macros, 45
mathematical functions, 16
operators, 15
AR20 through AR99, 52
ARG1 through AR19, 52
ARRAY
defined, 18
examples, 20
array parameters, 18
1-D table example, 26
2-D table example, 26
3-D table example, 26
ARRAY, 18
basics, 19
CHAR, 18
elements, 20
naming conflict, 23
defining, 22
editing interactively, 25
examples, 20
filling from a data file, 26
filling vectors, 25
interpolating values, 30
labeling, 43
listing, 22, 32
matrix operations, 37
operations, 35
plotting, 42
specifying values, 23
TABLE, 18, 21
elements, 21
writing data files, 33
Array parameters
ARRAY
elements, 20
*ASK command, 14
defined, 63
C
*CFCLOS command, 48
*CFOPEN command, 33, 48
*CFWRITE command, 14, 48
CHAR
defined, 18
examples, 20
limitation with *VEDIT, 25
naming conflict, 23
character parameters, 12
comment character, 15
*CREATE command, 47
create macro dialog box, 49
*CSET command, 64
*CYCLE command, 56
defined, 57
D
data descriptors, 14, 33
in messages, 66
data files
writing from arrays, 33
/DECRYPT command, 72
macros
introduction to programming, 45
*DIM command, 15, 22–23
*DO command, 56
defined, 57
do-loops, 56
vector operations, 35
do-while, 57
E
*ELSE command, 54, 66
defined, 57
*ELSEIF command, 14, 54, 66
defined, 57
/ENCRYPT command
defined, 71
encrypting macros, 71
*END command, 47
*ENDDO command, 56
defined, 57
*ENDIF command, 54, 66
defined, 57
/EOF command, 50
ETABLE command, 8
*EXIT command, 56
  defined, 57
expressions, 15

F
FILE command, 13
/FILENAME command, 13
files
  abbreviations, 5
Format
  data descriptors, 33
functions, 16

G
*GET command, 15
  assigning parameters, 10
GET functions, 10
global encryption key, 72
*GO command, 53, 56
/GOPR command, 56, 71
GUI
  interfacing with, 63

H
home directory, 47

I
*IF command, 14, 56–57, 66
  defined, 54
/INPUT command, 13, 45

L
Array parameters
  CHAR
    limitation with *VEDIT, 25
login directory, 47
looping, 56–57

M
macros
  control functions, 53
    quick reference, 57
creating, 46
creating status bar, 67
creating STOP button, 67
creating with a text editor, 49
displaying messages, 66
encrypting, 71
executing, 51
encrypted, 72
general examples, 61
library files, 50
local variables, 52
naming, 46
nesting, 45, 53
passing arguments to, 52
picking, 68
prompting for single parameter, 63
prompting with dialog box, 64
search path, 47
writing to session log, 68
matrix operations, 37
messages
  types of, 66
*MFouri command, 37
*MFun command, 37
*MOper command, 37
  examples, 37
*MSG command, 14
  defined, 66
MULTIPRO command
  defined, 64
  example, 64

N
/NOPR command, 71

O
operators, 15
  order of evaluation, 15
/OUTPUT command, 13

P
parameters, 7, 45
  _RETURN, 67
  _RETURN parameter, 59
  _STATUS parameter, 59
array (see array parameters)
assigning ANSYS-supplied values, 10
assigning at startup, 9
assigning during execution, 9
assigning through ANSYS command line, 9
assigning through ANSYS Launcher, 9
character, 12, 15
  defining, 9
deleting, 12
dynamic substitution of, 15
forcing substitution of, 13
listing, 11
maximum number, 11
PASSWORD, 72  
preventing substitution of, 13  
resuming, 17  
saving, 17  
substituting numeric values, 13  
using *GET, 10  
using get functions, 10  
using in-line get functions, 10  
writing, 17  

Parameters  
naming conventions, 8  
parametric expressions, 15  
parametric functions, 16  
PARRES command, 14, 17  
PARSAV command, 14, 17  
passing arguments to macros, 52  
plotting  
array vectors, 42  
labeling, 43  

/PMACRO command, 68  

R  
*REPEAT command, 56  
repeating a command, 56  
RESUME command, 13  
parameters  
retrieving or restoring parameter values, 31  

S  
*SET command, 9, 15, 23  
specification commands  
vector and matrix operations, 39  
start.ans file, 9  
status bar, 67  
*STATUS command, 11, 15, 32, 39  
examples, 32  
/STITLE command, 13, 15  
STOP button  
creating using a macro, 67  

T  
TABLE  
defined, 18  
examples, 21  
/TEE command, 48  
text editor, 49  
/TITLE command, 13, 15  
/TLABEL command, 13, 15  
toolbar  
default buttons, 3  
modifying, 3  

nesting abbreviations on, 5  
*TREAD command, 23, 26  

U  
UIDL functions  
calling in macros, 69  
/UIS command, 66  
*ULIB command, 51  
UNIX shells, 9  
unknown command  
method for executing macros, 51  
*USE command, 51  

V  
*VABS command, 39  
*VCOL command, 39  
*VCUM command, 39  
vector operations, 35  
*VEDIT command, 15, 23  
defined, 25  
*VFECT command, 39  
example, 39  
*VFILL command, 15, 23, 35  

defined, 25, 39  
*VFUN command, 15, 35  
examples, 35, 39  
*VGET command, 15, 35  
*VITRP command, 15, 35  
*VLEN command, 15, 39  
examples, 39  
*VMASK command, 15, 39  
example, 39  
*VOPER command, 15, 35  
examples, 35  
gather and scatter, 35  
*VPLT command  
defined, 42–43  
*VPUT command  
defined, 31  
*VREAD command, 14–15, 23, 26, 35, 39  
*VSCFUN command, 15, 35  
*VSTAT command, 39  
*VWRITE command, 14–15, 17, 35, 39  
data descriptors, 33  

W  
windows  
current directory, 47