

AIRA

APREP

Advanced Preprocessor for Scripting Languages

Reference Manual
Revision MAR2000

Airborne Research Australia

Australia's National Research Aircraft Facility at Flinders University



Contributors

APREP is the FORTRAN 90 successor of the FORTRAN 77 program PREP which was originally programmed by Alastair Williams and Scott Chambers during the late 1980s and early 1990s. The port to FORTRAN 90 and the conversion to a stand-alone program were done by Wolfgang Loeff in 1999. Many extensions to the original PREP functions were introduced by Sigrid Hacker in the late 1990s.

The current development team are:

Sigrid Hacker - programmer
Cécilia Ewenz - tester
Jörg Hacker - tester
Wolfgang Loeff - tester
Stuart Matthews - tester

Copyright

Copyright © 1989-2000 by Airborne Research Australia. All rights reserved worldwide

Trademarks

All brand and product names mentioned in this manual are trademarks or registered trademarks of their respective holders.

Disclaimer

Airborne Research Australia reserves the right to revise its software and publications with no obligation to notify any person or any organization of such revision. In no event shall Airborne Research Australia be liable for any loss of profit or any other commercial damage, including but not limited to special, consequential, or other damages.

**Airborne Research Australia
PO Box 335
Salisbury South, SA 5106
Australia**

**Phone: +61-8-8182-4000
Fax: +61-8-8285-6710**

<http://ara.es.flinders.edu.au>

1 Table of Contents

1	TABLE OF CONTENTS	3
2	INTRODUCTION	4
3	INVOCATION OF APREP	4
4	GENERAL COMMAND SYNTAX	4
5	ALPHABETICAL REFERENCE	5

2 Introduction

Processing of time series data, regardless of the nature of the data sampled, often involves typing highly repetitive sequences of commands, as the same (or only slightly different) processing techniques are applied to many subsets of the entire data set. When dealing with aircraft data for example, the same basic processing steps are typically applied to each of a large number of segments of the entire flight, corresponding to different geographical locations, heights, or other categorisations. In order to avoid the need for long script files containing many repetitions of large blocks of very similar command sequences, the script pre-processor APREP can be used.

This program takes a plain text file containing a mixture of scripting commands for the target processing program and specialised pre-processor abbreviation commands and labels, and "expands" it into the form of a "plain" script file.

3 Invocation of APREP

Under normal circumstances, APREP will nowadays be invoked automatically by the respective front/end of the processing system used, but if need be, it can also be executed from a command line prompt using the following syntax:

```
APREP [input file],[definition file],[output file]
```

or the abbreviated form

```
APREP [input file],[output file]
```

with the following parameters:

<i>[input file]</i>	name of the input file to process
<i>[definition file]</i>	name of a file which includes default settings to be initialized before any other processing steps are taken (optional)
<i>[output file]</i>	the file the resulting output should be written to

4 General command syntax

The preprocessor syntax can be grouped into two categories:

- a) commands and operators - these are prefixed with a '\$' and are **not case sensitive**.
- b) variables - these are enclosed in curly brackets '{...}' and are **case sensitive**.

5 Alphabetical Reference

[text1] ...
[text2]

When ... is used to terminate a line, the contents of the next line is appended to the characters immediately preceding the ... and processed as if it would be a single line.

[text1] ! *[text2]*

If an input line contains a !, it, and all following characters of that line are ignored (i.e. not interpreted any further and not copied into the output file). In order to be able to generate output lines containing exclamation marks, the \$! command is necessary.

Parameters:

[text1] any characters preceding the ! are fully processed

[text2] any characters following the ! are ignored

\$@S(*[variable reference]*,*[count]*)

This statement will be replaced by the n-th space-delimited substring of the referenced variable as indicated by *[count]*.

\$@U(*[text]*)

This statement will be replaced by the upper case equivalent of *[text]*.

[text1] \$! *[text2]*

If \$! is used at the beginning of the line, the normal function of the ! character is disabled for the remainder of that line. If \$! is used within a line, this pair of characters is ignored without affecting the interpretation of any other parts of that line.

\$& *[var]* *[exp1]* *[exp2]* . . . *[expn]*

Loop element extension line - multiple statements of this type can be included immediately after a \$LOOP command in order to add further variables to be loop-substituted simultaneously with the variable specified in the \$LOOP statement.

Parameters:

[var] the name of the loop variable

[exp1...expn] the list of required substitutions

Notes:

- i. Each set of replacement expressions for the loop variables must contain the same number of strings.
- ii. The *[exp*i*]* may be delimited by blanks or single quotes. Thus, if the user wishes one or more of *[exp*i*]* to contain blanks, then those replacement strings must be delimited by quotes. Quotes will be removed upon expansion.
- iii. *[var]* cannot contain blanks.

[expression1] **\$AND** *[expression2]*

Logical operator: yields 'true' if both expressions are true.

\$C *[line]*

A **\$C** at the beginning of a line will prevent the remainder of this line from being modified in any way by APREP - after the leading **\$C** is removed it is directly copied to the output file.

\$ENDLOOP

Endpoint of a **\$LOOP** construct

[string1] **\$EQ** *[string2]*

Relational operator: yields 'true' if both strings are fully identical in character and case.

\$IF *[expression]*

Evaluates the controlling expression for the conditional execution of **\$USE**, **\$THEN**, **\$ELSE**, and **\$ELSEIF** statements.

Parameters:

[expression] an expression formed by strings, relational operators (**\$EQ**, **\$NE**, and **\$IN**), and logical operators **\$OR** and **\$AND**) that can be evaluated while the line is being processed (i.e. the value of all variables used must be known). For details refer to Annex A.

[string1] **\$IN** *[string2]*

Relational operator: yields 'true' if *[string1]* is a substring of *[string2]*.

\$INCLUDE [*filename*] [*p1*] [*p2*] . . . [*pn*]

Replace the \$INCLUDE line by the contents of the specified file. The optional parameters [*p1*] to [*pn*] can be evaluated by a \$SUBSTITUTE command at the beginning of this file.

Parameters:

[*filename*] file to include
[*p1*] . . . [*pn*] optional parameters to pass to \$SUBSTITUTE

Note:

Pre-processor variables defined in the calling cmd-files prior to the \$INCLUDE statement remain valid within the include-file and subsequently.

\$INDEX [*ind*]

This statement can be included **once** after any \$LOOP/\$& statements, immediately prior to any further statements, and acts to initialise a variable {*ind*} which will contain the integer representation of the loop count.

Parameter:

[*ind*] the name of the variable for indexing

\$LOOP [*var*] [*exp1*] [*exp2*] . . . [*expn*]

Loops through a group of commands a number of times, whilst substituting the specified variable successively with different replacement strings. The loop construct has to be closed by an \$ENDLOOP statement.

Parameters:

[*var*] the name of the loop variable
[*exp1* . . . *expn*] the list of required substitutions

Notes:

- i. \$LOOP commands must have corresponding \$ENDLOOP commands.
- ii. Loops may be nested.
- iii. Loops may not contain \$SET statements.
- iv. The [*exp_i*] may be delimited by blanks or single quotes. Thus, if the user wishes one or more of [*exp_i*] to contain blanks, then those replacement strings must be delimited by quotes. Quotes will be removed upon expansion.
- v. [*var*] cannot contain blanks.

[string1] **\$NE** *[string2]*

Relational operator: yields 'true' if both strings are not fully identical in character and case.

[expression1] **\$OR** *[expression2]*

Logical operator: yields 'true' if at least one of the expressions is true.

\$SET *[name]* *[text]*

Assigns a character string to replace every occurrence of a variable. This will result in a global replacement of each occurrence of *{ [name] }* in a file with the contents of *text*, this will include any occurrences of *{ [name] }* prior to the **\$SET** statement.

Parameters:

[name] name of variable

[text] value to assign to variable

Notes:

- i. **\$SET** statements should be specified at or near the top of the input file. All **\$SET** statements must be specified before any **\$LOOP** or **\$IF** statements are specified. (It is not possible to specify **\$SET** statements inside **\$LOOP** or **\$IF** statements).
- ii. The **\$SET** statement does not recognise quotes as having special meaning. Thus, quotes will be regarded as part of the string to be transposed.
- iii. *[text]* is defined from the first non-blank character after 'name', to the last non-blank character of the line. Thus *[text]* may contain blanks.
- iv. String replacement in each line of the cmd-file will be performed in the same order as the **\$SET** statements appear in the cmd-file (significant when embedding variables)
- v. A **\$SET** statement cannot occur twice with the same variable name.
- vi. Both *[name]* and *[text]* are case sensitive.
- vii. **\$SET** statements themselves may contain variable refernces, provided these variables have been defined prior to their usage.
- viii. The maximum length of *[text]* after expansion of all variables is 200 characters.

\$SUBSTITUTE *[p1]* *[p2]* . . . *[pn]*

Causes the local symbols *[p1]* to *[pn]* to be replaced by the values defined by a calling **\$INCLUDE** command

Parameters:

[p1] . . . *[pn]* local symbols to be replaced

Notes:

- i. This command can only be used as the first command of an include file
- ii. Variable currently being used as `$LOOP` names must not be overridden by a `$SUBSTITUTE` name.

expression(i)
(<string1> RelOp <string2>) LogOp (<string3> RelOp <string4>) . . .

string(i)

are character strings, pairs of which will be compared by relational operator(s) "RelOp". The result of these comparisons are logicals which are then, in turn, compared pairwise by "LogOp" (a logical operator). The use of a logical operator is optional, and as such the smallest valid "expression" is a purely relational expression of the form: <string1> RelOp <string2>

RelOp

Currently valid relational operators are:

LogOp

Currently valid logical operators are:

The use of brackets to delimit operations is optionally allowed, and recommended since it improves the readability of the command, and can also be used to change the order by which expressions are evaluated. The following two statements are operationally identical: \$IF ((a \$EQ b) \$OR (a \$EQ c)) \$THEN . . . \$IF a \$EQ b \$OR a \$EQ c \$THEN . . .

"string1", "string2" etc, are delimited via the placement of the operators, and are assumed to start and end with non-blank characters. They may contain internal blanks, however. Quotes are not recognised as delimiters (and so will be regarded as part of the string).

(iii)

Any of "string(i)" or parts thereof, may be "{}"-variables previously set via \$SET statements or \$LOOP / \$& / \$INDEX commands.

(iv)

"string(i)" are case sensitive. The following expression would evaluate FALSE:
\$IF ARAmf \$EQ ARAmf . . .

(v)

The \$ELSEIF / \$ELSE commands are optional.

(vi)

\$IF commands can appear within loops.

(vii)

\$IF commands can be nested. (ie: Multiple-line \$IFs may contain any type or number of other \$IF statements, but Single-line \$IFs must not contain any other \$IFs or \$LOOPS).

(viii)

\$SET statements are not allowed within \$IF statements.

(ix)

The operators \$EQ, \$NE and \$IN must have a space on either side.

(x)

Operators are evaluated in the order \$EQ, \$NE, \$AND, \$OR, unless brackets are used, in which case expressions within brackets are evaluated first.

EXAMPLES:

\$IF statements are most commonly used in conjunction with either \$SET, or \$LOOP variables. The character expressions of these "variables" can be modified by the user, and compared to another string at stages throughout the cmd -file execution.

The Single Line \$IF Statement:

If it were necessary to have one series (say latitude) Reversed, the following would be a valid

usage of the one-line \$IF command: (first use the \$SET command to make a string variable)

```
$SET revLat Y  
$IF {revLat} $EQ Y $USE Reverse * latitude
```

The Multiple Line \$IF Statement:

If it were only necessary to have the latitude series Reversed on a particular day or flight of an experiment, the following is a valid use of the multiple-line \$IF command;

```
$SET day 2  
$IF {day} $EQ 2 $THEN  
  Reverse >newlat latitude  
$ELSE  
  newlat = <latitude  
$ENDIF
```