MAX-PLANCK-INSTITUT FÜR PLASMAPHYSIK

GARCHING BEI MÜNCHEN

R/32 Jan 1980

ACE - the AMOS Context Editor

- VERSION 1 -

Friedrich Hertweck
Ute Schneider

This document describes the philosophy and the specification of the AMOS Context Editor. ACE may be used to generate and modify labeled source file segments or unlabeled card image file segments, or to list or search for strings in print file segments produced by formatted FORTRAN WRITE statements.

Die nachstehende Arbeit wurde im Rahmen des Vertrages zwischen dem Max-Planck-Institut für Plasmaphysik und der Europäischen Atomgemeinschaft über die Zusammenarbeit auf dem Gebiete der Plasmaphysik durchgeführt.

Copyright © 1980 by
Max-Planck-Institut fuer Plasmaphysik
8046 Garching, GERMANY
Alle Rechte,
auch die des photomechanischen Nachdrucks, vorbehalten
All Rights Reserved

0. Introduction

The AMOS Context Editor ACE is designed to create, display, and modify S- or C-files or to display the contents of P-files or search for strings in P-files. S-files are AMOS file segments of records of 72 bytes of text plus a numeric label in columns 73:80 ("source" records), C-files are unlabeled "card image" file segments of 80 byte records of text, and P-files are file segments of up to 132 bytes of text, preceded by an ASA printer control character (i.e. records produced by FORTRAN formatted WRITE statements or job output spooled via remote station 13).

The ACE editor only contains the commands to achieve the above goal. The NUMBER command may be used to convert S-files (source files of labeled 72-byte records) into C-files and vice versa.

1. General Considerations

The context editor has two modes of operation:

(a) read-only mode (initiated by AMOS command DISPLAY), in which only commands are permitted that display the contents of the file segment; in this mode the user may access lines of the file segment in random order or he may search for lines containing (or not containing) a given string;

(b) update mode (initiated by AMOS command ACE), in which all commands may be used, especially those modifying the file segment; in this mode the file segment is processed sequentially from beginning to end, i.e. with increasing line numbers or labels.

When in update mode, a "current line" is activated by positioning commands. It may be modified or deleted, and new lines may be inserted right after the current line.

However, no line preceding the current line may be modified (but the update process may be repeated). After the update of the current line is completed, it is "pushed up" and the next line from "below" (i.e. from not yet used lines of the "old" file) is made the current line.

For the convenience of the user, there is a "display window" consisting of the 20 lines preceding the current line (provided there are that many lines preceding the current line), and the current line. The contents of this window may be listed at any time during the update process thus enabling the user to review actions on the file.

If during the update process a line above the window is

indicated in a LIST command or a line above the current line is indicated in a command intended to modify the line, the current update process is automatically completed by ACE, thereby producing a new version of the file segment. Then a new update cycle is started and the line indicated in the command is made the current line. This process may be described as a "wrap around".

1.1 Invoking the AMOS Context Editor

The AMOS Context Editor ACE runs under the AMOS System as a subsystem of the AMOS File Editor. After a user has signed-on to the AMOS System, he may invoke the AMOS Context Editor with either of the two commands:

ACE <segment designator>
DISPLAY <segment designator>

The response is a status display, including the file type, the sizes of LFIELD and SFIELD, the string delimiter, etc. and finally the word "ready", followed by the number of lines contained in the file segment. If the segment does not exist, then a new segment of type C-file is created by the ACE command. If the user inadvertantly created a segment by typing a wrong name, he may delete it using the FORGET command.

1.2 Prompting and ACE Responses

There are two prompt modes in ACE

- prompt for ACE commands with the symbol > (angle bracket) in the first position of the line
- prompt for data input without a prompt symbol such that 80 characters may be typed in a line.

Because data input mode is ended by empty input, a blank line must be input by at least one blank character followed by a carriage return.

ACE responses other than the display of lines of the file segment being processed will always start with the symbol ! in column 1.

1.3 Command Format

All ACE commands consist of a command verb (or a one letter abbreviation) followed by one or more operands which can be

- a line range
- a line selector
- a change specification

Only the operand syntax of the FIND command (which has no verb) and that of the control and status commands differ from that format.

More than one command can be put on an input line if the commands are separated by semicolons:

```
<command list>  ::=  <command>
                   |  <command list> ; <command>
```

A command input line may not exceed 80 characters (including the prompt symbol >).

2. ACE Commands

Normal operation of ACE is in command mode in which the user is prompted for input by an >, usually after the current line has been displayed:

```
{contents of current line}
>
```

The user may now enter one of several kinds of ACE commands:

- display and search commands (which in DISPLAY mode may access lines in random order)
- commands that access the current line alone
- line replacement commands
- string replacement (CHANGE) command
- DELETE command
- control and status commands
- END and OFF commands

The syntactic description of the commands follows the general rules of BNF notation. The complete syntax specification is attached with Appendix 1.

Vocabulary

In the description below, the command verbs are displayed in "full length, upper case" form. It is understood that for each command verb the lower case form and a standard abbreviation are also valid. For instance the possible forms for a LIST command are:

```
LIST   L   list   l
```

The following special symbols are used in the syntax of ACE:

```
+  -  *  \  |  ,  .  ;  :  !  ?  =  "  '  (  )  [ ]  @  #  &
```

Strings

```
<string> ::= <pattern string>
           | <word string>
```

Certain commands search for the occurrence of specified strings of characters. A <pattern string> is any series of legal (i.e. ASCII printable) characters. A <word string> is a <pattern string> which is delimited in the text by special characters (including line begin and line end) or blanks. This distinction is useful as a means of referencing variables in a source program. For example it allows finding all occurrences of the FORTRAN variable A without interferences from occurrences of the letter A in the variable ALPHA or the command READ.

Strings must be enclosed between a pair of string delimiters which must not appear in the string. The closing string delimiter must always be the same as the opening string delimiter.

The chosen string delimiter denotes whether a <pattern string> or a <word string> is the subject of the search.

ACE uses three string delimiters: a primary string delimiter " and an alternate string delimiter ' for pattern strings, and # for word strings. The latter two may be changed by the user (cf. SET command) to any of the symbols

$ _ / # '

The Field Parameter

With the field parameter

j:k

the columns j, j+1, ..., k can be selected for display, search, or modification. There are global field parameters to control the character positions in the line ("card columns") for listing and searching, and some commands may have their individual local field indication.

2.1 Display and Search Commands

The display and search commands can be used for all types of file segments: S-, C-, and P-files. If the command DISPLAY had been used to activate the file segment, it cannot be modified.

LIST

All lines specified by the parameters are displayed at the terminal. The current line pointer remains unchanged.

The LIST command has the syntax

```
<list>  ::=  LIST <lines>
           |  LIST <range>
           |  LIST <-range>
           |  LIST <selector>
           |  LIST

<lines>  ::=  <num> : <num>
           |  <num> : *
           |  = <num>
           |  *
           |  * - <num>
           |  * - <num> : *

<range>  ::=  + <num>
           |  <num>

<-range>  ::=  - <num>
            |  - <num> : <num>

<selector>  ::=  <sel> <window>
              |  @ <num>

<sel>       ::=  <expr>

<expr>      ::=  <term>
               |  <expr>  |  <term>

<term>      ::=  <factor>
               |  <term>  &  <factor>

<factor>    ::=  <primary>
               |  \  <primary>

<primary>   ::=  <pattern string>
               |  <word string>
               |  ( <expr> )

<window>  ::=  <range> <field>
             |  <range>
             |  <field>
             |

<field>  ::=  <num> : <num>
```

The <lines> specification may only be used in read-only mode because the numbers specify absolute line numbers (not "card labels"!) which are meaningless in an update environment (an insert or delete effectively changes all the line numbers below the current line). The construct =<num> denotes the line with line number <num>, the constructs with a colon a range of lines, and the asterisk the last line of the file segment.

The <range>, if empty, specifies the current line; the form +<num> specifies the sequence of lines x,...,x+<num> where x is the current line number; the form <num> specifies the lines x+1, ..., x+<num>. The <-range> specifies lines from the "display window" plus possibly some lines

"down-file".

The <selector> is used to select those lines from a range for which the boolean expression <sel> is true. The truth value of <pattern string> and <word string> is determined by their occurrence in the line. From there on, the normal rules for evaluating boolean expressions are applied (& denoting the AND operator, | denoting the OR operator). The empty string (i.e. "") may be used to select all lines from a range. The search for the strings involved in <sel> is performed in the indicated range of lines, possibly restricted by <field> which specifies the columns in which the search is to be performed. An empty range specification means all remaining lines of the file segment, an empty field specification means all columns of a line.

The selector @<num> is used for S-files to denote the label with numerical value <num>; it is equivalent to the selector #<num># 73:80. The search is terminated when the indicated line is found or at end-of-file.

FIND

The FIND command locates the line for which the selector has the value "true". The line found is made the current line and displayed. The FIND command has the syntax

```
<find>  ::=  <line spec>
           |  <selector>

<line spec>  ::=  = <num>
                |  + <num>
                |  - <num>
                |  +
                |  -
                |  *
                |  * - <num>
```

If a line specification =<num> is given, the line is directly located. The specification +<num> indicates the line x + <num>, where x is the current line pointer; - <num> denotes the line x - <num>. The symbol + stands for +1 and the symbol - for -1.

The asterisk denotes the last line of the file segment.

The selector indicates a search for the given string. The search starts with the first line following the current line and continues till the range is exhausted or the indicated string is found, or end-of-file is reached. The line containing the string is made the current line.

If the string was not found, the response is "!not found" and the last line of the range is made the current line.

2.2 Current Line Commands

One way of updating a file segment is by proceeding through the file segment with FIND commands and replacing, modifying or deleting the current line, or inserting new lines right after the current line. Typing an = causes a copy of the current line to be inserted (i.e. the current line is duplicated). The difference between this mode of updating and the update cycle controlled by the "line replacement" commands described in the next section is that only the current line is accessed by the command.

The commands to be used are

REPLACE
MODIFY
ALTER
DELETE
INSERT

all without parameters, thus indicating the current line as operand.

The first three commands put the ACE processor into replace, modify, or alter mode, respectively. The subsequent input is treated as a replacement line, a modification directive, or a string alteration directive, respectively. The format of these directives is described below. After these commands have been given, ACE stays in the indicated mode, thus permitting the current line to be updated repeatedly. Updating of the current line is terminated by empty input or by a single / or % sign. Entering a single letter command "r", "m", or "a" will change the current mode to the new mode.

During modification the current line CL is extended by an auxiliary line AL of the same length. The valid linelength is determined by the type of the file segment (i.e. 80 for C-file and 72 for S-file segments). If an input line IL is longer than CL, the AL receives the overflowing characters. The contents of the auxiliary line can be made the subject of further modification(s) described below.

If at the end of an update process the auxiliary line contains non-blank characters, it is converted into a normal line and inserted right after the current line (and appropriately labeled in case of an S-file).

Modification directives

When modification of the current line is requested (by typing in a single "M"), the current line is extended by an auxiliary line of the same length containing all blanks.

The current line and the auxiliary line will be treated as one entity in the modification process. Any characters inserted in the current line will cause the remaining characters to be shifted to the right. Characters shifted from the current line will be saved in the auxiliary line, shifting the contents of the auxiliary line to the right.

Any characters shifted from the auxiliary line will be lost. If characters are deleted from the current line, the reverse actions take place by supplying additional characters at the end of the current line from the auxiliary line, the latter being padded with blanks in turn.

Note that the modification directive acts on both the current line and the auxiliary line. For instance, if the input line is longer than 72 characters for an S-file, the remaining input is used to modify the auxiliary line.

During the modification process the current line CL and the input line IL are matched character by character. The character in the input line determines what has to be done and what character is to be placed into the result line RL.

The following actions are performed if the input line contains one of the indicated items:

blank (copy)
the CL character above the blank is moved to RL;

<string> (insert)
the string is moved to RL, then the characters of CL above <string> (including the characters above the string quotes) are copied to RL;

_ (delete)
the CL character above the underscore _ is skipped (and will therefore appear to be deleted from RL);

| (blank-out)
the CL character above the stroke | is skipped and a blank character is moved to RL;

! (split)
extend RL with blanks; the characters in CL above ! and all characters to the right of the ! sign (including the auxiliary line) are shifted to the auxiliary line (any non-blank characters shifted from the auxiliary line are lost);

!! (truncate)
the characters in CL above the first ! and all following characters are deleted;

& (concatenate auxiliary line)
the CL character above & and all remaining characters are skipped (i.e. they will appear to be deleted from RL); the characters from the auxiliary line are copied to RL; if the auxiliary line is blank, no concatenation takes place;

&& (catenate auxiliary or next line)
the CL character above the first & and the characters following it are replaced by: the auxiliary line if it is non-blank, or the next line from the old file segment if the auxiliary line is blank (any overflowing characters will be placed into the auxiliary line);

!& (delete auxiliary line, catenate next line)
the contents of the auxiliary line are discarded; the CL character above the & and the characters following it are replaced by the next line from the old file segment (any overflowing characters will be placed into the auxiliary line);

any other characters (replace)
the CL character is skipped, the input character is moved to RL.

^ (activate auxiliary line)
the auxiliary line is made the current line; this command may be used to access the auxiliary line, for instance for modification (Note: on the Siemens terminals the symbol @ must be used);

\ (restore)
a \ as single character input restores the previous version of the current and auxiliary lines.

Modification of a line is terminated by entering empty input or a / or % sign. In the first case ACE will return to command mode and select the next line from an update cycle; in the latter cases the update cycle is terminated and the user is prompted by a > sign.

String alteration directive

The current line may also be modified by the ALTER command. Subsequent input must be a

<string pair> ::= "<sequence of char>"<sequence of char>"

where the character sequence must not contain the string quotes. Instead of the " the alternate string delimiters or the word string delimiters may be used. In the current line all occurrences of the first string are replaced by the second string.

Delete current line

The DELETE command displays and deletes the current line, followed by the message "!DELETED" and a command prompt > .

Insert new lines

The INSERT command places ACE into insert mode. Subsequent input lines cause the current line to be pushed up; the input line then becomes the current line. Thus a sequence of new lines may be inserted right after the current line.

Typing a = causes a copy of the current line to be inserted (i.e. the current line is duplicated).

Typing in one of the letters "r", "m", or "a" permits the last line that was inserted to be replaced, modified, or altered, respectively. Typing an "i" returns to insert mode.

The insert mode normally is terminated by empty input. Instead, a / or % may be used.

Typing a "d" will delete the last line inserted.

2.3 Line Replacement Commands

The line replacement commands are used to replace or modify a sequence of lines specified by a selector. Alternatively, the lines selected may be deleted, or additional lines may be inserted after the selected lines. The command given to intitialize the replacement loop (REPLACE, MODIFY, or ALTER) or the delete or insertion loop sets the primary mode, thus establishing the rules for the interpretation of the user's input. However, it is possible to switch to any secondary mode, thus permitting the user complete freedom of how to handle his file segment.

REPLACE, MODIFY, ALTER, DELETE, or INSERT loops

These commands are used to initialize the primary replacement mode to replace a sequence of existing lines by other lines (either typed in or produced by modifying the existing lines). Some lines may also be left unchanged, others may be deleted or additional lines may be inserted.

All these commands set up a loop of update operations over the lines specified by the selector. The syntax of the commands is:

```
<replace>  ::=  REPLACE <selector>

 <modify>  ::=  MODIFY <selector>

  <alter>  ::=  ALTER <selector>

 <delete>  ::=  DELETE <selector>

 <insert>  ::=  INSERT <selector>
```

One by one, ACE makes each line selected from the range the current line, displays it, and goes into input mode. The user may now enter

- in replace mode:
  a new line which replaces the current line;

- in modify mode:
  a modification directive (cf. 2.2) to modify the current line;

- in alter mode:
  an alteration directive to replace in the current line all occurrences of the first string by the second string.

These inputs may be entered repeatedly, thus modifying the current line again and again. If the user inputs

- in insert mode:
  a new line, it is inserted after the current line which may be the one previously inserted.

Entering

- empty input:
  if not in delete mode empty input will push the current line up and make the next line selected from the range the current line; if in delete mode the current line is deleted, the message !DELETED is displayed and the next line is selected and displayed;

However, the user may also switch into a secondary line replacement mode by entering a single character

a to switch to alter mode for the current line; subsequent input must be an alteration directive

m to switch to modify mode for the current line; subsequent input must be a modification directive

r to switch to replace mode for the current line; subsequent input is treated as a replacement line

d to delete the current line, display the message !DELETED, select the next current line and display it, at the same time returning to the primary line replacement mode

i to switch to line input mode to insert new lines right after the current line; the current line is pushed up and the input line is made the current line

= to push the current line up, make an exact copy of it which becomes the current line (duplication of current line) and switch to modify mode, thus permitting the duplicate to be modified.

Whenever an empty input is given in secondary replacement mode, the primary replacement mode is re-established and the next current line is selected and displayed.

Finally, the input of

/ or

% terminates the input cycle before the range is exhausted; the last line handled remains the current line; note that in a DELETE cycle the current line is then not deleted.

2.4 Delete a Contiguous Sequence of Lines

With another form of the DELETE command a sequence of contiguous lines can be deleted from the file segment. The syntax is:

```
<delete>  ::=  DELETE <range> <go>
            |  DELETE <sel-range> <go>
```

```
<sel-range>  ::=  <selector> :: <selector>

    <go>  ::=  empty
           |   !
           |   ! !
```

The <range> specifies a contiguous sequence of lines that are to be deleted. <sel-range> also specifies a contiguous set of lines to be deleted, but the first is found with the first selector in the specified window, the last line is found with the second selector, starting from the next line (i.e. the line following the one found by the first selector).

If <go> is empty the changes are performed in "verify mode": if the range contains less than four lines, all of them are listed; if the range contains more than 3 lines, the first and the last lines of the range are listed and the user is prompted to reply:

empty input effectuates the delete

\ as input cancels the command

If <go> is ! the user is not prompted but the deleted lines are listed (as described above); if <go> is !! the command is performed without prompting the user and without listing anything.

2.5 String Replacement Command

The string replacement command is used to change a string where it occurs in a range of lines into another string (which may be empty). Optionally, the search for and replacement of the string may be restricted to a <field> of columns of the line range. The syntax is:

```
<change>  ::=  CHANGE <rep> <string pair> <window> <go>
   <rep>  ::=  <num> *
           |   empty
```

In all lines of the range the first string, in the order of appearance, is replaced <num> times by the second string. If the <field> specification of <window> is empty, the search is performed in the whole line. If <rep> is empty, all occurrences of the first string are replaced.

If <go> is empty the changes are performed in "verify mode": for each line in which the search string is found, the current line as it was before the change and the changed line are displayed; then the secondary update mode is set to "modify" and the user is prompted for a modification directive:

empty input confirms the change

\ as input restores the original version of the line

a modification directive permits arbitrarily other modifications

input of the single letters a, r, i, or d permits any other desired modifications.

If <go> is ! the lines are displayed as described above; the user is not prompted but ACE continues to find the next line. If <go> is !! the display is suppressed also.

3. Line Variables, Procedures, and GOTO

In this section the more advanced concepts of line variables and procedures are introduced. They serve to automate update processes further.

3.1 Line Variables

A set of line variables V(1), V(2),..., V(19) may be used to store lines that are used repeatedly. Each line variable may hold a line of up to 80 characters (depending on the file type).

There are two ways to specify the contents of a line variable or a sequence of line variables:

```
<v def> ::= <v range> =
          | <v range> = X

<v range> ::= V ( <num> )
            | V ( <num> : <num> )
```

The first form of <v range> specifies just one line, the second form a contiguous sequence of lines.

If the first form of <v def> is used, then ACE goes into input mode and the specified number of lines may be entered. After the last line has been input, ACE returns into command prompt mode.

In the second form of <v def>, the line variables are filled with the current line and the lines following the current line. The lines used are displayed.

The command

```
<v display> ::= <v range> ?
```

may be used to display the current contents of the specified line variables.

The line variables may be used whenever line input is

requested from the user (i.e. for REPLACE or INSERT commands, string modification, or alteration directives).

As an example, the command sequence

```
V(1:7)=X; D+6; "DO 12  K=1,N"; I V(1:7)
```

moves seven lines to another place further down in the file (inserting an additional "=0;" after the DELETE moves the lines up).

In general, <v range> may be used in any command where input is requested. A range of more than one line may only be used for insertions.

If <go> is !, no user response is necessary to proceed; if <go> is !!, in addition listing is restricted or suppressed. The syntax is:

```
<command> ::= REPLACE <selector> <v range> <go>
            | INSERT  <selector> <v range> <go>
            | MODIFY  <selector> <v range> <go>
            | ALTER   <selector> <v range> <go>
```

3.2 Command Procedures

The user may define procedures which consist of one or more command lines. Each command line may hold up to 80 characters. They are denoted by P(1), P(2),...,P(19). They are defined in the same way as line variables:

```
<p def> ::= <p range> =
          | <p range> = X

<p range> ::= P ( <num> )
            | P ( <num> : <num> )
```

The first form of <p range> specifies one command line, the second form a contiguous sequence of command lines.

The specified command lines may be used similarly to procedures in an ordinary programming language:

```
<p call> ::= <p>
           | <num> * <p>

<p> ::= P ( <num> )
      | P . <id>
```

The second form of <p call> causes the procedure to be executed <num> times. In <p> the first line of "procedure" is indicated. The second form is only used for predefined library procedures where the identifier denotes the procedure to be used (yet to be implemented).

If during procedure execution an error message or a warning is displayed, it is preceded by the display of the current procedure command line. In some cases after the message a command prompt is made with the prompt symbol

being an ! to indicate that the procedure execution is still in process; then a command input can terminate procedure execution.

Whenever during procedure execution input from the terminal is requested, a single % sign serves as escape character and stops procedure execution immediately.

The command

<p display> ::= <p range> ?

may be used to display the current contents of the specified procedure lines.

Procedures or line variables are valid during a terminal session as long as only the AMOS commands ACE or DISPLAY are used; other AMOS commands may destroy the contents of procedures or line variables.

3.3 GOTO Command

If a procedure consists of more than one command line, the lines must be concatenated by GOTO commands:

<go to> ::= GOTO <num>

This command continues execution with procedure line P(<num>), i.e. it should be the last command on a command line.

If <num> in a GOTO command is zero, the current command line is repeated. It is the only form of GOTO that may be used outside procedure execution, i.e. it applies to the command line that has been explicitly typed in.

Procedures may be nested up to 4 levels deep. Procedure execution may be terminated by typing in a % whenever input is requested. ACE then returns to command prompt mode.

3.4 Remarks

In order not to loose track of the state of procedure execution, the user may use the remark command:

<remark> ::= [<sequence of characters>]

Whenever this command is encountered, it is printed including the brackets [...].

4. Control and Status Commands

These commands are used to set certain environmental parameters and to display them.

4.1 SET Command

The SET command is used to set the parameters LFIELD (list field), SFIELD (search field), the boolean variables LABEL and TEXT, and the alternate delimiters:

```
        <set>  ::=  SET <set list>

   <set list>  ::=  <set parm>
                 |  <set list> , <set parm>

   <set parm>  ::=  <set field>
                 |  <set bool>
                 |  <set del>

   <set bool>  ::=  <bool var>
                 |  \ <bool var>

   <bool var>  ::=  TEXT
                 |  LABEL

  <set field>  ::=  <field var> = <num> : <num>
                 |  <field var>

  <field var>  ::=  LFIELD
                 |  SFIELD

    <set del>  ::=  DEL = <">
                 |  WDEL = <">
```

<set bool> assigns the value "true" or "false" to the boolean variable, respectively; the second form of <set field> resets the field to the default values:

	LFIELD	SFIELD
S-file	1: 80	1: 72
C-file	1: 80	1: 80
P-file	2:133	1:133

The default settings differ slightly for the different types of terminal devices (linelength etc.).

The <set del> sets the alternate string delimiter to any of the characters " $ _ / # and ' . The characters ' and # are the preset default values for pattern strings and word strings, respectively.

4.2 NUMBER Command

This control command is used to convert C-files into S-files, and vice versa. The syntax is:

```
      <number>  ::=  NUMBER <sequence>
                  |  NUMBER

    <sequence>  ::=  <num> . <num>
                  |  . <num>
```

If the command is used with <sequence>, then the first form of <sequence> is used to convert a C-file into an S-file, or to renumber an S-file. The command must be given when the file segment has just been opened or after a FIND command of the form "=0" has been issued (i.e. the file segment has been positioned at its beginning, just before the first line). Numbering is then done by proceeding through the file segment.
The second form is used to set the starting value for the label increment when inserting lines into an S-file. The default value is 10, but it is gradually decreased, if necessary, to 1 depending on the label of the next line and the number of lines being inserted.

If NUMBER is used without an operand, then for both C- and S-files columns 73:80 are set to blank and an S-file becomes a C-file.

4.3 QUERY and X Commands

These commands are used to display the environmental parameters (QUERY) and the current line pointer (X); the syntax is:

```
<query>  ::=  QUERY
           |  X
```

4.4 END and OFF Commands

The END command completes the update process (if the ACE command had been invoked) and returns to the AMOS line editor. The file segment that has been used remains active. The parameter TEXT retains its value. The syntax is:

```
<END> ::= END
```

The OFF command performs the same actions as END and invokes the usual AMOS OFF in addition. The syntax is:

```
<OFF> ::= OFF
```

4.5 FORGET Command

The FORGET command discards all updates made since the last backward positioning of the current line, i.e. -n, where n>0, or =n, where n < current line pointer. In other words, during an update process, the user steps through his file segment, leaving modified lines "above" the current line. The ACE system produces the new version of the file segment during this process. The FORGET command simply discards the updated version of the file segment and positions the "old" file segment at the beginning. The syntax is:

```
<FORGET> ::= FORGET
```

If a new file segment was created via the ACE command,

the FORGET command discards the file segment as a whole, i.e. the file segment is purged and the session continues with the AMOS line editor.

4.6 Transition to the AMOS Line Editor

In the command prompt mode (indicated by the > symbol) the symbol % followed by an AMOS Line Editor command or blanks completes the ACE update or display process. If a command was specified, it is transferred to the AMOS Line Editor and immediately executed.

Appendix 1: ACE Command Syntax

In the following summary of the ACE command syntax, the commands are grouped according to their purpose and the page where they appear in the text is indicated.

General Rules and List Commands

```
                                                          Page
<command list> ::= <command>                               3
                 | <command list> ; <command>

<string> ::= <pattern string>                               4
           | <word string>

<list>  ::= LIST <lines>                                    5
          | LIST <range>
          | LIST <-range>
          | LIST <selector>
          | LIST

<lines>  ::=  <num> : <num>                                 5
           |  <num> : *
           |  = <num>
           |  *
           |  * - <num>
           |  * - <num> : *

<range>  ::=  + <num>                                       5
           |  <num>

<-range>  ::=  - <num>                                      5
            |  - <num> : <num>

<selector>  ::=  <sel> <window>                             5
              |  @ <num>

<sel>        ::=  <expr>                                    5

<expr>       ::=  <term>                                    5
              |   <expr> | <term>

<term>       ::=  <factor>                                  5
              |   <term> & <factor>

<factor>     ::=  <primary>                                 5
              |   \ <primary>

<primary>    ::=  <pattern string>                          5
              |   <word string>
              |   ( <expr> )

<window>  ::=  <range> <field>                              5
            |  <range>
            |  <field>
            |

<field>  ::=  <num> : <num>                                 5
```

```
<p def> ::= <p range> =                                          14
          | <p range> = X

<p range> ::= P ( <num> )                                        14
            | P ( <num> : <num> )

<p call> ::= <p>                                                 14
           | <num> * <p>

<p> ::= P ( <num> )                                              14
      | P . <id>

<p display> ::= <p range> ?                                      15

<go to> ::= GOTO <num>                                           15

<remark> ::= [ <sequence of characters> ]                        15
```



```
      <set>  ::=  SET  <set list>                                15

<set list>  ::=  <set parm>                                      16
               |  <set list> , <set parm>

<set parm>  ::=  <set field>                                     16
               |  <set bool>
               |  <set del>

<set bool>  ::=  <bool var>                                      16
               |  \ <bool var>

<bool var>  ::=  TEXT                                            16
               |  LABEL

<set field>  ::=  <field var> = <num> : <num>                     16
               |  <field var>

<field var>  ::=  LFIELD                                         16
               |  SFIELD

  <set del>  ::=  DEL = <">                                      16
               |  WDEL = <">

   <number>  ::=  NUMBER <sequence>                              16
               |  NUMBER

 <sequence>  ::=  <num> . <num>                                  16
               |  . <num>

    <query>  ::=  QUERY                                          17
               |  X

      <end>  ::=  END                                            17
      <off>  ::=  OFF                                            17
   <forget>  ::=  FORGET                                         17
```