USER'S MANUAL FOR APAREL:
A PARSE-REQUEST LANGUAGE

R. M. Balzer

PREPARED FOR:
ADVANCED RESEARCH PROJECTS AGENCY
MEMORANDUM
RM-6104-ARPA
JANUARY 1970

USER'S MANUAL FOR APAREL:
A PARSE-REQUEST LANGUAGE
R. M. Balzer

This research is supported by the Advanced Research Projects Agency under Contract No. DAHC15-67-C-0111. Views or conclusions contained in this study should not be interpreted as representing the official opinion or policy of ARPA.

DISTRIBUTION STATEMENT
This document has been approved for public release and sale; its distribution is unlimited.
PREFACE

This Memorandum describes the use of APAREL, a parsing capability embedded within the PL/I language. The APAREL extension allows users to specify both the syntax of their parse-requests in a BNF-like language and the semantics associated with a successful parse-request in the PL/I language.

The Memorandum is based on the assumption that the reader has read APAREL--A Parse-Request Language† and that he understands the basic ideas of top-down parsing.

APAREL has been developed as a basic tool for use in man-machine communication studies at The RAND Corporation under the sponsorship of the Advanced Research Projects Agency.

This Memorandum is a user's manual for APAREL, which is a parse-request language. It describes the features that have and have not been implemented, the restrictions on the use of these facilities, the new features added to APAREL since the publication of *APAREL--A Parse-Request Language*, the method of invoking the available facilities, and ideas on the effective and efficient use of APAREL.

I. INTRODUCTION

APAREL is presently implemented as a set of subroutines callable from PL/I. Therefore, APAREL programs must be set up using these calls rather than those specified in *APAREL--A Parse-Request Language* [1]. In addition, certain features mentioned in that publication have not yet been implemented, while certain new features have been added. Also, several implementation restrictions exist. All of the above are detailed in this manual.

The Memorandum is based on the assumption that the reader has read *APAREL--A Parse-Request Language* [1], and that he understands the basic ideas of top-down parsing.
II. USE OF APAREL

All parsing capabilities of APAREL are invoked by calls to the APAREL parser. These calls are used in the following ways: 1) to define, redefine, and delete parse-requests, 2) to define parse-related names, 3) to initiate a parse-request, 4) to terminate the semantics of a parse-request, and 5) to turn the trace of the parsing on or off.

Each call can be given at any time, with the exception of terminating the semantics of a parse-request, which can only be issued from a semantic routine initiated by APAREL as the result of a successful parse-request. Hence, except as noted above, all APAREL functions can be dynamically invoked, providing such features as:

1) Dynamic addition of new parse-requests;
2) Dynamic redefinition of parse-requests;
3) Dynamic tracing of parse;
4) Recursive initiation of parse-requests.

However, since no incremental compiler is available for PL/I, semantic routines cannot be dynamically added, redefined, or deleted. The routines are:

1) DEFINE_PARSE_REQUEST

This routine is used to define or redefine a parse-request or a parse-related name (such as a semantic-routine name). If the parse-request or parse-related name specified has already been defined, it is deleted and defined as if it were new.

This routine has three or four arguments. First is the parse-request being defined, which is passed as a character string. Its form is as specified in APAREL--A Parse-Request Language [1], except the double colons at each end are not present. The second argument is a character string into which the results of a successful parse of that parse-request will be placed. Third is a binary fixed variable into which
the number of the successful option of that parse-request will be placed. The fourth argument, if present, is a label in the PL/I program to which control will be passed upon successful completion of the parse-request; i.e., it is the label at the start of the semantic routine for the parse-request. (The routine TERMINATE_PARSE, as explained below, terminates the semantics of a parse-request.) If the first argument consists of only a single name or a single name followed by a colon, it is interpreted as the definition of a parse-related name. Its use in other parse-requests determines its type of parse-related name. These types are:

a) **PARSE_NAME**: If the name appears followed by a colon, it is interpreted as being a local parse name. The parse results and the parse-results option (as specified in the second and third arguments, respectively) of the call that defined the parse-related name will be set to the parse results of the PARSE_ALTERNATIVE_GROUP in which the PARSE-NAME appeared. A fourth argument, if specified in the defining call, will be initiated as the semantic routine for the parse-related name.

b) **PARSE_TIME_ROUTINE_NAME**: If the name appears after a semicolon in a parse-request, it is interpreted as being the name of a parse-time semantic routine. The label specified as the fourth argument in the call defining the parse-related name will be initiated as a semantic routine.

c) **Indirect parse specification**: If neither above condition holds, the parse-related name is treated as the indirect specification of a parse rule, and the current value of the second argument in the defining call of the parse-related name is used as the invoked parse-request.
2) PARSE

This routine, which is used to initiate a parse-request, has three arguments, each of which is a character string. The first is the input string; i.e., the string to be parsed. This string will not be altered by APAREL. The value of the second argument is the parse-request that will be used to parse the input. It can be a complex parse-request or, as is usually the case, simply the name of a previously defined parse-request; it is used merely to invoke that parse-request. The third argument is a character string into which will be placed that portion of the input string that was not parsed successfully. During the parsing of the original parse-request, if any parse-request (the original, any initiated by it, or any they initiate, etc.) is successful and has a semantic routine specified or if a PARSE_TIME_ROUTINE_NAME is encountered, the parse is temporarily suspended, and the semantic routine is initiated. After it returns (see TERMINATE_PARSE below), the parse is resumed.

3) TERMINATE_PARSE

This routine returns control to APAREL from a semantic routine; it has one argument, a binary fixed value. If the value is zero (unsuccessful), APAREL will continue the parse as if the current parse-request had syntactically failed at the current point (further alternatives may still allow the parse-request to be successful). If the value is nonzero (successful), the parse will continue as if the semantics had not been invoked. In either case, if the semantic routine alters the value of the parse results (the second argument in the DEFINE PARSE REQUEST routine), the altered value will be passed to any higher-level parse-requests and used in forming their parse results.
4) DELETE_PARSE_REQUEST

This routine deletes a parse-request; it has one argument—a character string—which is the name of the parse-request to delete.

5) TRACE_PARSE

This routine, used to turn tracing on or off, has no arguments. Each call changes the setting of the trace switch from off to on, or vice versa.

6) COMPILE_PARSE_REQUEST

This routine defines a parse-request just as the define-parse-request routine does; it has the same arguments with the same usage. This routine is used when all alternatives are one character literals and when the parse-request is used frequently. Instead of testing each alternative sequentially until a successful one is found or until all have been tried, this routine builds a translate table [2] to test all alternatives simultaneously in parallel; hence, the speed of the parse is greatly improved.
III. ADDITION AND OMISSION OF FEATURES

The following features have been omitted in the present implementation of APAREL:

1) The BAL function—string balanced with respect to specified arguments.

2) PARSE-REQUEST-SEQUENCES—the user must set up a parse-request that contains, as alternatives, the desired sequence of parse-requests; e.g., if the parse-request sequence A1, A2, A3, A4 is desired, the call

\[
\text{PARSE(input, 'A1|A2|A3|A4', remaining_input)}
\]

will effect the parse-request sequence.

3) INPUT and OUTPUT VARIABLES.

4) The NORMAL SEPARATION and SEMANTICS OPEN or CLOSED statements.

The following features have been added:

1) Ability to redefine parse-requests dynamically through the DEFINEPARSE_REQUEST routine.

2) Ability to trace a parse-request dynamically.

3) A NOT function—it can be stated in a parse-request that the input must not match a particular PARSE_ELEMENT (specified by the NOT symbol (~), followed by the PARSE_ELEMENT not wanted). If the PARSE_ELEMENT is successful, the alternative will fail; if the PARSE_ELEMENT is unsuccessful, the parsing of the alternative will continue. For example, in a language with reserved words, and assuming a parse-request called RESERVED_WORD exists to define these words, the definition of an identifier might be

\[
\text{identifier:=-reserved_word letter(-ARBNO(alphanumeric,-))}
\]
That is, an identifier is a letter followed optionally by an arbitrary number of alphanumeric separated by NULLs (and with no intervening blanks as specified by the minus signs), which is not a reserved word.

Similarly, to define a relation as an arbitrary number of terms separated by relational operations, but including at least one relational operator (i.e., a single term is not to be a relation), the following parse-request can be used:

\[
\text{relation} : \neg \langle \text{term}-\text{relational\_operator} \rangle \\
\text{ARBNO} (\text{term}, \text{relational\_operator})
\]

4) A termination function—this function, specified by the slash symbol (/) in a parse-request and used to require that the end of the input string be reached, is successful if no nonblank characters remain unparsed in the input string. Furthermore, if the termination function is preceded by a minus sign (-), it will be successful only if the entire input string has been parsed; i.e., no characters remain unparsed.
IV. IMPLEMENTATION RESTRICTIONS

1) A parse-request cannot have more than four nested levels within it. Each nested PARSE_GROUP (a set of alternatives enclosed in '⟨' and '⟩' brackets) or ARBNO function counts as one level. Thus, the parse-request

\[ B: ⟨C|D|E \text{ARBNO(F,G)}⟩|H⟩(I|J) \]

has three nested levels (two PARSE_GROUPS and the ARBNO function).

2) A PARSE_RESULT's maximum size is 256 bytes.

3) The maximum number of elements in a parse-request, counting one for each PARSE_ATOM, PARSE_NAME, and APAREL syntax operator ('⟨', '⟩', '∣', '.', etc.), must not exceed 128.

4) The total number of PARSE_REQUESTS, PARSE_RELATED names, and unique literals within PARSE_REQUESTS must not exceed 2048.

5) PARSE_ALTERNATIVE_NAMES cannot be used.

6) PARSE_TIME routines (specified within a parse-request following a semicolon) must have no parameters.

7) All semantic routines must be in the same PL/I block as the call to PARSE, which initiated the parse-request invoking the semantic routines.

8) The ARB function may not appear inside an ARBNO function immediately. It can be used inside an ARBNO function if it is a PARSE_ATOM, which is part of a PARSE_GROUP (i.e., it is enclosed in a pair of '⟨' '⟩' brackets). Thus,

\[ \text{ARBNO}(⟨A|\text{ARB }B⟩|C),D⟩ \]

is acceptable.
9) Normal separation is assumed to be zero. Hence, if one or more blanks are desired, the period notation must be used.

10) A PARSE_NAME cannot be specified for the ARB function. For any other PARSE_ATOM, this can be accomplished by preceding the PARSE_ATOM with a parse name and enclosing the pair in PARSE_GROUP brackets (e.g., (name: atom)). This method of naming (via the right-angle bracket) ends the parse group in which the parse atom occurs and, as explained in Sec. V, prevents the ARB function from working correctly.
V. PROGRAMMING CONSIDERATIONS

To use APAREL effectively, the user should be aware of its basic method of parsing. The two types of backup in parsing are: 1) when the input pointer backs up as mismatches are encountered, and 2) when the PARSE_RULE (or its equivalent) pointer backs up. APAREL uses only the first of these; i.e., the PARSE_RULE pointer moves strictly left to right through a parse rule (two exceptions are explained below). Within a PARSE_ALTERNATIVE_GROUP, each alternative is tried until one is found that is successful (e.g., in the parse-request

NAME: (A1|A2|A3)B1|B2

A2 is successful). Then the parser skips to the end of the PARSE_ALTERNATIVE list (the bracket after A3) and processes the next PARSE_ELEMENT (B1), if any, in the PARSE_ELEMENT_LIST. If this PARSE_ELEMENT (B1) fails, the parser will again skip to the next alternative (B2) in that PARSE_ALTERNATIVE_LIST. It will not go back and try alternative A3 followed by B1; thus, the ordering of alternatives in a PARSE_REQUEST is important. If one of two alternatives can match a prefix of the input that the other can match, the second alternative should be placed before the first in a PARSE_ALTERNATIVE_LIST; e.g., the alternatives A1 and A1 A2 should be ordered

A1 A2|A1

in a PARSE_REQUEST. The "longest" or "biggest" alternatives should be placed first.

The ARBNO and ARB functions are the two exceptions to the strict left to right movement of the PARSE_RULE pointers. The ARBNO function matches an arbitrary but nonzero number
of occurrences of the first argument; these occurrences are separated by occurrences of the second argument. The PARSE_REQUEST pointer will alternate between these arguments until one fails (if the first argument fails, the input pointer is backed up past the last occurrence of the second argument). The PARSE_REQUEST pointer will then skip past the right parenthesis after the second argument.

The ARB function, which matches an arbitrary string, matches first a string of zero length, and the parse_request pointer moves to the next PARSE_ELEMENT in the PARSE_ELEMENT list (e.g., in the PARSE_REQUEST

NAME: A1 ARB A2(B1|B2)A3|A4

this would be A2). If this PARSE_ELEMENT or any further one (say A3) in the PARSE_ELEMENT_LIST fails, the PARSE_REQUEST pointer is backed up to the ARB; the length of the string that the ARB matches is increased by one; and the PARSE_REQUEST pointer again moves to the next PARSE_ELEMENT (A2) in the PARSE_ELEMENT_LIST. This process is repeated until either the entire PARSE_ELEMENT_LIST succeeds or until the ARB runs out of input to match, in which case the PARSE_ELEMENT_LIST fails. In either case, processing continues, as with normal PARSE_ELEMENT lists.

Left-recursion is handled uniquely. The state of the parser is determined by two variables: 1) the position in the input string, and 2) the position in the parse-request. Before attempting a match for any alternative, the parser checks to see if the present state has occurred before (during the current initiation of the original parse-request). If it has, a left recursive loop has occurred and the parser simply moves on to the next alternative to break the left recursive loop. Therefore, this would cause the rule

number: number digit|digit
to fail on more than two-digit numbers. This can be remedied by using the ARBNO function, which allows iterative specification rather than nested recursive definition; thus,

number: ARBNO(digit,-)

A number is an arbitrary nonzero number of digits separated by NULLs (the minus sign ensures that no embedded blanks are in the number); or, even more elegantly:

expression: ARBNO(expression,operator)|(expression)
            |variable|number
            |unary_operator expression

An expression is an arbitrary nonzero number of expressions separated by operators, a parenthesized expression, a variable, a number, or a UNARY_OPERATOR followed by an expression.

Care also must be exercised with semantic routines, those specified as PARSE_TIME semantics, or those specified as semantic routines for PARSE_REQUESTS. After they are invoked and have returned, as the parse continues, the input for which they were invoked may be backed up past. It may then be reparsed or it may remain as part of the unparsed input. For example, in the rule

Variable: identifier '(', ARBNO(expression,'',''),')' | identifier

(a variable may be either a subscripted or an unsubscripted identifier), assuming that 'identifier' has a normal definition and that a semantic routine is specified for it, 'identifier' will be invoked twice if the input string consists of an unsubscripted identifier. Both times it is invoked for the same parsed result (the identifier in the input), the first time as part of subscripted identifier.
After the first invocation of the semantic routine has returned, the first alternative will fail because a left parenthesis will not be found. The input pointer will be backed up past the identifier in the input stream, and the second alternative will be tried. The identifier will be reparsed, and the semantic routine reinvoked.

To avoid this problem, the PARSE_REQUEST can be given as:

```
Variable: identifier('('ARBNO(expression,','))')
```

(A variable is an identifier followed optionally by a subscript.) Here the identifier is parsed only once.

When using the minus sign (meaning no blanks may be between two PARSE_ATOMS) as the last element in the separator (second argument) of an ARBNO function, care must be used if the repetition string (first argument of ARBNO) has alternatives. The minus sign would apply to the first alternative in the repetition string since it always applies to only the next PARSE_ELEMENT. Normally, the minus sign is meant to apply to each alternative; this can be accomplished by enclosing the repetition string in angle brackets (thus making it a PARSE_GROUP and, hence, a single PARSE_ELEMENT).
VI. OPTIMIZATION

The user can do several things to speed up the parse and to reduce the amount of space it requires.

A) When defining a heavily used PARSE_REQUEST consisting of only one-character literals (e.g., the definition of 'letter'), use the routine COMPILE_PARSE_REQUEST rather than DEFINE_PARSE_REQUEST. This causes a translate table to be built and allows the alternatives to be tested in parallel simultaneously. This can greatly affect efficiency.

B) When specifying one alternative that is a prefix or a suffix of another, factor out the common portion and specify the rest as an option. For example,

\[ A_1 \rightarrow A_2 \rightarrow A_3 | A_1 A_2 \]

should be specified instead as

\[ A_1 \rightarrow A_2 \rightarrow (A_3 | ) \]

This is especially important if \( A_1, A_2, \) or both are complex PARSE_REQUESTS as it can save extensive reparsing.

C) When possible, nested recursive (either left or right) definitions of parse-requests should be changed to iterative, or iterative recursive, definitions. For example, instead of defining number as:

\[ \text{Number: } \text{Number digit|digit} \quad \text{(left recursive)} \]

or as:

\[ \text{Number: } \text{digit Number|digit} \quad \text{(right recursive)}, \]

\[ \text{Number: } \text{Number digit|digit} \quad \text{(left recursive)} \]
it can be defined as:

\[
\text{Number: ARBNO(digit,)} \quad \text{(iterative definition)}.
\]

D) Finally, the ARB function should not be used more than is necessary since its use may involve large amounts of reparsing.
TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */

TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */
/* THIS PROGRAM PROVIDES AN ON-LINE SYNTAX CHECKER, IT ALLOWS THE USER TO SPECIFY HIS SYNTAX IN APAREL FORMAT AND TO TEST THIS SYNTAX AGAINST INPUT HE SUPPLIES ON-LINE. THIS PROGRAM IS ITSELF WRITTEN IN APAREL AND USES APAREL TO PARSE THE USERS COMMANDS ITS SEMANTIC ROUTINES THEN MAKE USE OF APAREL AGAIN TO DEFINE OR REDEFINE A RULE SPECIFIED BY THE USER- OR TO TEST A RULE 'IN SOME INPUT SUPPLIED BY THE USER. THE ON-LINE INTERACTION IS SUPPLIED BY A SET OF PL/I CALLABLE SUBROUTINES(SUPPLIED BY DICK WEKELBLATT OF BELL LABS) WHICH INTERFACE WITH THE IBM 2260 ALPHANUMERIC DISPLAY UNIT. */

/* THESE CALLS TO APAREL DEFINE THE SYNTAX LANGUAGE USED BY THIS SYNTAX-TESTER. THE PARSE_REQUEST_NAMES ARE PURPOSELY CHOOSED LONG SO THAT THE USER WILL NOT INADVERTANLY REDEFINE THEN WHEN DEFINING HIS OWN LANGUAGE. */


CALL DEFINE_PARSE_REQUEST('TEST_SYNTAX_NAME' TEST_SYNTAX_LETTER-CAR8NOCTEST_SYNTAX_LETTER TEST_SYNTAX_DIGIT-12', 'SIMPLE_VARIABLE', 'SIMPLE_VARIABLE_OPTION');

CALL COMPILE_PARSE_REQUEST('TEST_SYNTAX_LETTER', 'ABCDIEFGHIJKLMNOPQRSTUVWXYZ', 'LETTER', 'LETTER_OPTION');

CALL COMPILE_PARSE_REQUEST('TEST_SYNTAX_DIGIT', '01123456789', 'DIGIT', 'DIGIT_OPTION');

CALL DEFINE_PARSE_REQUEST('PARSE_RULE_NAME', 'NIL', 'PARSE_RULE_NAME_OPTION');
/* THESE PARSE_REQUEST DEFINITIONS ARE INCLUDED SO THAT THE ON-LINE USER NEED NOT DEFINE THESE COMMONLY USED PARSE_REQUESTS UNLESS HE WISHES TO MAKE A NON-STANDARD DEFINITION. */

DECLARE 'IVARIABLE; SUBSCRIPT; SIMPLE_VARIABLE; ALPHANUMERIC.
BOOLEAN_EXPRESSION; RELATIONAL_OPERATOR; LOGICAL_OPERATOR; NUMBER; LETTER; DIGIT; EXPRESSION; UNARY_OPERATOR; OPERATOR);
TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */

CHARACTER(10) VARYING,
(VARIABLE_OPTION, SUBSCRIPT_OPTION, SIMPLE_VARIABLE_OPTION,
BOOLEAN_EXPRESSION_OPTION, RELATIONAL_OPERATOR_OPTION,
LOGICAL_OPERATOR_OPTION,
ALPHANUMERIC_OPTION, NUMBER_OPTION, LETTER_OPTION,
EXPRESSION_OPTION, UNARY_OPERATOR_OPTION, OPERATOR_OPTION,
DIGIT_OPTION) BINARY FIXED,
NULL CHARACTER(0) EXTERNAL,
SPECIFIED BINARY FIXED INITIAL(1),
UNSPECIFIED BINARY FIXED INITIAL(0),
SUCCESSFULLY BINARY FIXED INITIAL(1),
UNSUCCESSFULLY BINARY FIXED INITIAL(0);

CALL DEFINE_PARSE_REQUEST(
  'VARIABLE: SIMPLE_VARIABLE <
   **(** ARBN0(SUBSCRIPT: EXPRESSION>,***) )***) | >>,
  VARIABLE, VARIABLE_OPTION);

CALL DEFINE_PARSE_REQUEST(
  'SUBSCRIPT: SUBSCRIPT, SUBSCRIPT_OPTION);

CALL DEFINE_PARSE_REQUEST(
  'SIMPLE_VARIABLE: LETTER< ARBN0(ALPHANUMERIC,-) | >>,
  SIMPLE_VARIABLE, SIMPLE_VARIABLE_OPTION);

CALL DEFINE_PARSE_REQUEST(
  'ALPHANUMERIC: LETTER | DIGIT',
  ALPHANUMERIC_ALPHANUMERIC_OPTION);

CALL DEFINE_PARSE_REQUEST(
  'NUMBER: ARBN0(DIGIT,-)',
  NUMBER, NUMBER_OPTION);

CALL Compile_PARSE_REQUEST(
  LETTER, LETTER_OPTION);

CALL DEFINEPARSE_REQUEST(
  'DIGIT: 0|1|2|3|4|5|6|7|8|9',
  DIGIT, DIGIT_OPTION);

CALL DEFINE_PARSE_REQUEST(
  'EXPRESSION: ARBN0(EXPRESSION, OPERATOR)
  | VARIABLE | NUMBER | **(** EXPRESSION **)')
  | UNARY_OPERATOR EXPRESSION * EXPRESSION,
  EXPRESSION_OPTION);

CALL DEFINE_PARSE_REQUEST(
  'OPERATOR: * | ** | * | **/** | **/** | ****,
  OPERATOR, OPERATOR_OPTION);

CALL DEFINE_PARSE_REQUEST(
  'UNARY_OPERATOR: * | ** | **/** | **/** | UNARY_OPERATOR,
  UNARY_OPERATOR_OPTION);

CALL DEFINE_PARSE_REQUEST
TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */

*BOOLEA_ Expres_sion: AExp(BOOLEA_ Expres_sion,LOGICAL_OPERATOR)
  | | BOOLEA_ Expres_sion | | EXPRES_sion RELATIONAL_OPERATOR
BOOLEA_ Expres_sion,BOOLEA_ Expres_sion,OPTION; 19
CALL DEFINE_PARSE_REQUEST;
  *RELATIONAL_OPERATOR: *** | *** | = | | > | | <**
  | ***->** | ***<**
  RELATIONAL_OPERATOR,RELATIONAL_OPERATOR,OPTION;
20
CALL DEFINE_PARSE_REQUEST;
  *LOGICAL_OPERATOR: **| ** | **| **
  LOGICAL_OPERATOR,LOGICAL_OPERATOR,OPTION;
21
UNSPE_IC(START_SYMBOL)='010D1010'B;
22
OPEN_SCOPE:
  CALL OOPEN('SCOPE ');
23
  /* OPEN THE USER TERMINAL FOR INTERACTION */
INIT_IALIZE_SCREEN:
  CALL GWRI_TE(EWL,UNIT,*CAPAREL ONLINE SYNTAX TESTER READY.*);
24
  /* ERASE THE SCREEN & WRITE A LINE(EWL) ON LINE : (THE
   FIRST CHARACTER OF THE MESSAGE ) */
  WAIT:
    IF -GETEXI(TANY_UNIT) THEN /* INTERRUPT IS NOT PENDING FROM USER */
25
    CALL GWAITUNIT UNIT; /* WAIT FOR ANY INTERRUPT */
26
  READ_REQUEST:
  CALL GETJUSTI(UNIT,REQUEST); /* READ REQUEST */
27
  ECHO_REQUEST;
    MESSAGE=LINE(11111111REQUEST;
28
    CALL GWRI_TE(EWL,UNIT,MESSAGE);
29
    /* REWRITE THE INPUT REQUEST ON LINE 1 OF DISPLAY */
  DECODE_REQUEST:
    SIMPLE_VARIABLE='';
30
    CALL PARSE(REQUEST,'DEBUG_SYNTAX',REMAINING_INPUT);
    /* PARSE THE INPUT(REQUEST) USING THE PARSE REQUEST
    'DEBUG_SYNTAX' AND PUT THE REMAINING INPUT IN STRING
    REMAINING_INPUT */
31
    GO TO ROUTINE('DEBUG_SYNTAX,OPTION);
    /* USE DEBUG_SYNTAX,OPTION, SET BY THE PARSE INVOKED ABOVE
    AS AN N-WAY SWITCH TO GO TO THE PROPER SEMANTIC ROUTINE */
32
    ROUTINE(0); /* INVALID INPUT */
33
    MESSAGE='ILLEGAL INPUT FOR THE APAREL ONLINE SYNTAX TESTER.';
    GO TO PROCESSING_COMPLETE;
34
    ROUTINE(1); /* NEW PARSE RULE */
TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */

CALL PARSE(REMAINING_INPUT,'SIMPLE_VARIABLE',
REMAINING_INPUT2); /* GET PARSE_NAME */
35 CALL FIND_NAME; /* SEARCH FOR PARSE_NAME */
36 IF I=0 THEN DO;
/* NAME NOT FOUND. CREATE NEW PARSE RULE */
38 DO I=I TO PARSE_RULE_INDEX;
39 IF PARSE_RULE[I]="" THEN /* EMPTY SPACE IN TABLE FOUND */
40 GO TO DEFINE_RULE;
41 END;
42 PARSE_RULE_INDEX=PARSE_RULE_INDEX+1;
43 I=PARSE_RULE_INDEX;
44 DEFINE_RULE:
MESSAGE="NEW PARSE RULE ACCEPTED:'
45 END;
46 ELSE /* RULE ALREADY EXISTS, CHANGE IT */
MESSAGE="PARSE RULE 'SIMPLE_VARIABLE' HAS BEEN REDEFINED:'
47 PARSE_RULE[I]=REMAINING_INPUT;
48 PARSE_RESULT[I]=""
49 PARSE_RESULT_OPTION[I]=0;
50 CALL DEFINE_PARSE_REQUEST(
REMAINING_INPUT,PARSE_RESULT),
PARSE_RESULT_OPTION[I],
51 GO TO PROCESSING_COMPLETE;

ROUTINE(2); /* TEST INPUT FOR RULE */
52 CALL FIND_NAME; /* SEARCH FOR RULE NAME */
53 IF I=0 THEN DD; /* RULE FOUND */
55 IF TRACE=I THEN
56 CALL TRACE_PARSE; /* TRACE THIS RULE */
57 CALL PARSE(REMAINING_INPUT,SIMPLE_VARIABLE,REMAINING_INPUT2);
58 IF TRACE=I THEN DD;
59 CALL TRACE_PARSE; /* TURN TRACE BACK OFF */
60 PUT PAGE;
61 DO J=1 TO PARSE_RULE_INDEX;
63 PUT SKIP 2 LIST(*PARSE RULE*+PARSE_RULE[I]);
64 IF PARSE_RESULT_OPTION[I]=UNSPECIFIED THEN
MESSAGE=*PARSE RULE UNSUCCESSFUL;
66 ELSE
MESSAGE=*ALTERNATIVE!*PARSE_RESULT_OPTION[I]
SUCCESSFUL*;
67 PUT SKIP LISTMESSAGE;
68 PUT SKIP LIST(*PARSE RESULTS*+PARSE_RESULT[I]);
69 END;
70 PUT PAGE;
71 END;
72 IF PARSE_RESULT_OPTION[I]=UNSPECIFIED THEN
TEST_SYNTAX: PROCEDURE OPTIONS (MAIN); /* FILE TESTSYN */

73  MESSAGE='PARSE OF RULE SIMPLE_VARIABLE' UNSUCCESSFUL';
74  ELSE DO; /* PARSE WAS SUCCESSFUL */
75    MESSAGE=LINE(2) 'ALTERNATIVE' PARSE_RESULT_OPTION
76    ' SUCCESSFUL';
77    CALL WRITE_LINE_UNIT MESSAGE;
78    MESSAGE=LINE(3) 'PARSED INPUT' PARSE_RESULT;
79    CALL WRITE_LINE_UNIT MESSAGE;
80    MESSAGE='INPUT PARSED SUCCESSFULLY BY RULE SIMPLE_VARIABLE';
81    END;
82    END;
83    END;
84  ELSE /* PARSE RULE NOT FOUND */
85    RULE DOES_NOT_EXIST:
86    MESSAGE='PARSE RULE SIMPLE_VARIABLE DOES NOT EXIST';
87    GO TO PROCESSING_COMPLETE;
88
89  ROUTINE(3): /* TRACING PARSE */
90    TRACE=1-TRACE; /* FLIP TRACE SWITCH */
91    MESSAGE='TRACE OF PARSE SWITCH FLIPPED';
92    GO TO PROCESSING_COMPLETE;
93
94  ROUTINE(4): /* DISPLAY LIST OF PARSE RULES */
95    IF SIMPLE_VARIABLE=' ' THEN DO;
96      /* START DISPLAY FROM NAMED RULE */
97      CALL FIND_NAME; /* SEARCH FOR RULE */
98      IF I=0 THEN
99        PARSE RULE POSITION=1;
100      ELSE
101        GO TO RULE DOES NOT_EXIST;
102      END;
103    ELSE /* RULE NOT SPECIFIED, CONTINUE FROM PRESENT POSITION */
104      IF PARSE RULE POSITION PARSE RULE INDEX THEN
105        PARSE RULE POSITION=1; /* START OVER */
106      DU = 2 TO 10 WHILE (PARSE RULE POSITION PARSE RULE INDEX);
107      CALL WRITE_LINE_UNIT MESSAGE PARSE RULE;
108      PARSE RULE POSITION=1;
109      DU = 1 TO 10 WHILE LENGTH (PARSE RULE PARSE RULE POSITION)
110      IF 1=1 THEN
111        SKIPT getline
112      END;
113    END;
114    MESSAGE='CONSECUTIVE PARSE RULES DISPLAYED';
115    GO TO PROCESSING_COMPLETE;
TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */

106 ROUTINE(5): /* DISPLAY INDIVIDUAL PARSE RULE */
    CALL FIND_NAME; /* SEARCH FOR PARSE RULE */
    IF I=0 THEN GO TO RULE_DOES_NOT_EXIST;
    ELSE DO:
        /* PARSE RULE FOUND */
        CALL WRITELINE,UNIT,LINE(2)11*PARSE RULE**
        /* PARSE RULE(11) */
        IF PARSE_RESULT_OPTION(11)_UNSPECIFIED THEN
            MESSAGE=LINE(11)11*PARSE RULE ** SIMPLE_VARIABLE**
            /* UNSUCCESSFUL */
        ELSE /* PARSE WAS SUCCESSFUL */
            MESSAGE=LINE(11)11*ALTERNATIVE**PARSE_RESULT_OPTION(11)
            /* IF PARSE RULE ** SIMPLE_VARIABLE** SUCCESSFUL */
            CALL WRITELINE,UNIT,MESSAGE;
        END;
        CALL WRITELINE,UNIT.LINE(6)11*PARSE RESULTS**
        /* PARSE_RESULT(11) */
        MESSAGE=**PARSE RULE ** SIMPLE_VARIABLE** DISPLAYED.**
        GO TO PROCESSING_COMPLETE;
END;

119 ROUTINE(6): /* LIST */
    DO I=1 TO PARSE RULE_INDEX;
        IF PARSE RULE(11)** THEN
            PUT SKIP(2) LIST(PARSE RULE(11));
        END;
    END;
    PUT PAGE;
    MESSAGE=**ALL PARSE RULES LISTED.**
    GO TO PROCESSING_COMPLETE;

126 ROUTINE(7): /* PUNCH */
    DO I=1 TO PARSE RULE_INDEX;
        IF PARSE RULE(11)** THEN
            PUT FILE(PUNCH) SKIP(2) LIST(PARSE RULE(11));
        END;
    END;
    MESSAGE=**ALL PARSE RULES PUNCHED.**
    GO TO PROCESSING_COMPLETE;

132 ROUTINE(8): /* READ */
    ON ENDFILE(SYIN) GO TO PROCESSING_COMPLETE;
    MESSAGE=**PARSE RULES HAVE BEEN READ IN.**
    DO WHILE(IPARSE RULE_LEN*X<#BOUND(PARSE RULE,11));
            GET LIST(REMAINING_INPUT)
            CALL PARSE(REMAINING_INPUT,** SIMPLE_VARIABLE**.

133 134 135 136 137
TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */

REMAINING_INPUT2;
138   CALL FIND_NAME; /* SEARCH FOR PARSE NAME */
139   IF I=0 THEN DO;
140      /* RULE DOES NOT EXIST YET */
141      DO I=1 TO PARSE_RULE_INDEX;
142      IF PARSE_RULE(I)="" THEN GO TO DEFINE_RULE;
143   END;
144   PARSE_RULE_INDEX=PARSE_RULE_INDEX+1;
146   I=PARSE_RULE_INDEX;
147   END;
148   DEFINE_RULE1;
149      PARSE_RULE(I)=REMAINING_INPUT;
150      PARSE_RESULT(I)="";
151      PARSE_RESULT_OPTION(I)="";
152      CALL DEFINEPARSE_REQUESTPARSE_RULE(I);
153      MESSAGE="IMPLEMENTATION RESTRICTION: YOU HAVE TOO MANY RULES.";
154      GO TO PROCESSING_COMPLETE;
155
ROUTINE(99): /* DELETE */
156   CALL FIND_NAME;
157   IF I=0 THEN GO TO RULE DOES_NOT_EXIST;
158   CALL DELETE_PARSE_REQUEST(SIMPLE_VARIABLE);
159   PARSE_RULE(I)=""; /* INDICATE RULE DELETED */
160   MESSAGE="PARSE RULE \"SIMPLE_VARIABLE\" HAS BEEN DELETED."
161   GO TO PROCESSING_COMPLETE;
162
ROUTINE(101): /* STOP */
163   CALL GCLOSE;
164   UNSPECIFIED('0110101010';
164   CALL SGMESS\('128'\)|\'32',/* STOP*/FULL); /* ISSUE STOP
165   MESSAGE TO SG. SET CONTROL BYTE FOR INTERPRET MESSAGE
166   AS COMING FROM USER'S SCOPE, INPUT AND OUTPUT FROM THERE
167   ALSO /*
168   GO TO OPEN_SCOPE;
169
ROUTINE(111): /* CLEAR */
170   DO I=1 TO PARSE_RULE_INDEX;
171   CALL PARSEPARSE_RULE(I)="/SIMPLE_VARIABLE",REMAINING_INPUT1;
172   CALL DELETE_PARSE_REQUEST(SIMPLE_VARIABLE);
173   PARSE_RULE(I)="/" /* INDICATE PARSE RULE DELETED */
174   END;
TEST_SYNTAX: PROCEDURE OPTIONS(main); /* FILE TESTSYN */

171  PARSE_RULE_INDEX=3;
172  MESSAGE="# ALL PARSE RULES HAVE BEEN ULETEG.;";
173  GO TO PROCESSING_COMPLETE;

174  ROUTINE(12): /* FINISH */
      RETURN;

175  PROCESSING_COMPLETE:
      MESSAGE=LINE(111) MESSAGE;
      CALL WRITEFILE,UNIT,MESSAGE,START_SYMBOL);
      GO TO WAIT;

178  FIND_NAME: PROCEDURE:
179  REMAINING_INPUT2=PARSE_RULE_NAME;**;SIMPLE_VARIABLE;**;
180  /* CREATE PARSE RULE TO FIND NAME */
181  DO I=1 TO PARSE_RULE_INDEX;
182     CALL PARSE(PARSE_RULE(I1),REMAINING_INPUT2,REMAINING_INPUT3);
183     IF PARSE_RULE_NAME_OPTION=SPECIFIED THEN RETURN;
184   END;
185  I=0; /* PARSE RULE NOT FOUND */
186  END FIND_NAME:

187  DECLARE
      PUNCH FILE PRINT,
      REMAINING_INPUT CHARACTER(80) VARYING,
      REMAINING_INPUT CHARACTER(80) VARYING,
      ROUTINE(115) LABEL,
      PARSE_RULE_INDEX POSITION Binary Fixed Initial(1),
      SGNSNES ENTRY Binary Fixed(1),
      EDM CHARACTER(11),
      PARSE_RULE(10) CHARACTER(60) VARYING,
      PARSE_RULE_INDEX Binary Fixed Initial(1),
      DEBUG_SYNTAX OPTION Binary Fixed,
      PARSE_RULE_NAME_OPTION Binary Fixed,
      PARSE_RESULT(10) CHARACTER(133) VARYING,
      PARSE_RESULT OPTION(50) Binary Fixed,
      TRACE Binary Fixed Initial(1),
      I Binary Fixed,
      J Binary Fixed;

188  DECLARE
      GTEST RETURNS(Bit(11)),
      UNIT Binary Fixed Initial(1),
      WHI Binary Fixed Initial(1).
TEST_SYNTAX; PROCEDURE OPTIONS(IN); /* FILE TESTSYN */

RSB BINARY FIXED INITIAL(2),
LNE BINARY FIXED INITIAL(2),
EWL BINARY FIXED INITIAL(3),
ECHO_TYPE BINARY FIXED INITIAL(1),
ANY_UNIT BINARY FIXED INITIAL(255),
CLEAR BINARY FIXED INITIAL(256),
ONSCOPE BINARY FIXED INITIAL(0) EXTERNAL,
GCODE BINARY FIXED INITIAL(0) EXTERNAL,
GCOUNT BINARY FIXED INITIAL(0) EXTERNAL,
LINE1:11 CHARACTER(1) INITIAL("0","1","2","3","4","5","6","7","8","9","A","B"),
MESSAGE CHARACTER(1),
INITIAL_MESSAGE CHARACTER(1),
REQUEST CHARACTER(160) VARYING,
REMAINING_INPUT CHARACTER(160) VARYING,
START_SYMBOL CHARACTER(1),
WAIT_ENTRY(BINARY FIXED);

END TEST_SYNTAX;

189
<table>
<thead>
<tr>
<th>DCL NO.</th>
<th>IDENTIFIER</th>
<th>ATTRIBUTES AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>ALPHANUMERIC</td>
<td>AUTOMATIC, UNALIGNED, STRING(50), CHARACTER, VARYING</td>
</tr>
<tr>
<td>7</td>
<td>ALPHANUMERIC_OPTION</td>
<td>AUTOMATIC, ALIGNED, BINARY, FIXED(15, 0)</td>
</tr>
<tr>
<td>188</td>
<td>ANY_UNIT</td>
<td>AUTOMATIC, ALIGNED, INITIAL, BINARY, FIXED(15, 0)</td>
</tr>
<tr>
<td>7</td>
<td>BOOLEAN_EXPRESSION</td>
<td>AUTOMATIC, UNALIGNED, STRING(50), CHARACTER, VARYING</td>
</tr>
<tr>
<td>7</td>
<td>BOOLEAN_EXPRESSION_OPTION</td>
<td>AUTOMATIC, ALIGNED, BINARY, FIXED(15, 0)</td>
</tr>
<tr>
<td>188</td>
<td>CLEAR</td>
<td>AUTOMATIC, ALIGNED, INITIAL, BINARY, FIXED(15, 0)</td>
</tr>
<tr>
<td>187</td>
<td>COMPILE_PARSE_REQUEST</td>
<td>EXTERNAL, ENTRY, DECIMAL, FLOAT(SINGLE)</td>
</tr>
<tr>
<td>187</td>
<td>DEBUG_SYNTAX_OPTION</td>
<td>AUTOMATIC, ALIGNED, BINARY, FIXED(15, 0)</td>
</tr>
<tr>
<td>29</td>
<td>DECODE_REQUEST</td>
<td>STATEMENT LABEL CONSTANT</td>
</tr>
<tr>
<td>44</td>
<td>DEFINE_PARSE_REQUEST</td>
<td>EXTERNAL, ENTRY, DECIMAL, FLOAT(SINGLE)</td>
</tr>
<tr>
<td>44</td>
<td>DEFINE_RULE</td>
<td>STATEMENT LABEL CONSTANT</td>
</tr>
<tr>
<td>148</td>
<td>DEFINE_RULE1</td>
<td>STATEMENT LABEL CONSTANT</td>
</tr>
<tr>
<td>148</td>
<td>DELETE_PARSE_REQUEST</td>
<td>EXTERNAL, ENTRY, DECIMAL, FLOAT(SINGLE)</td>
</tr>
<tr>
<td>7</td>
<td>DIGIT</td>
<td>AUTOMATIC, UNALIGNED, STRING(50), CHARACTER, VARYING</td>
</tr>
<tr>
<td>7</td>
<td>DIGIT_OPTION</td>
<td>AUTOMATIC, ALIGNED, BINARY, FIXED(15, 0)</td>
</tr>
<tr>
<td>DCL NO.</td>
<td>IDENTIFIER</td>
<td>ATTRIBUTES AND REFERENCES</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>27</td>
<td>ECHO_REQUEST</td>
<td>STATEMENT LABEL CONSTANT</td>
</tr>
<tr>
<td>188</td>
<td>ECHO_TYPE</td>
<td>AUTOMATIC,A, INITIAL, BINARY, FIXED (15,0)</td>
</tr>
<tr>
<td>187</td>
<td>EOM</td>
<td>AUTOMATIC, UNALIGNED, STRING (1), CHARACTER</td>
</tr>
<tr>
<td>188</td>
<td>EWL</td>
<td>AUTOMATIC, A, INITIAL, BINARY, FIXED (15,0)</td>
</tr>
<tr>
<td>7</td>
<td>EXPRESSION</td>
<td>AUTOMATIC, UNALIGNED, STRING (50), CHARACTER, VARYING</td>
</tr>
<tr>
<td>7</td>
<td>EXPRESSION_OPTION</td>
<td>AUTOMATIC, A, BINARY, FIXED (15,0)</td>
</tr>
<tr>
<td>178</td>
<td>FIND_NAME</td>
<td>ENTRY, DECIMAL, FLOAT (SINGLE)</td>
</tr>
<tr>
<td></td>
<td>GCLOSE</td>
<td>EXTERNAL, ENTRY, DECIMAL, FLOAT (SINGLE)</td>
</tr>
<tr>
<td>188</td>
<td>GCODE</td>
<td>STATIC, EXTERNAL, A, INITIAL, BINARY, FIXED (15,0)</td>
</tr>
<tr>
<td>188</td>
<td>GCOUNT</td>
<td>STATIC, EXTERNAL, A, INITIAL, BINARY, FIXED (15,0)</td>
</tr>
<tr>
<td>188</td>
<td>GOPEN</td>
<td>EXTERNAL, ENTRY, DECIMAL, FLOAT (SINGLE)</td>
</tr>
<tr>
<td></td>
<td>GREADV</td>
<td>EXTERNAL, ENTRY, DECIMAL, FLOAT (SINGLE)</td>
</tr>
<tr>
<td>188</td>
<td>GTEST</td>
<td>EXTERNAL, ENTRY, STRING (1), 8</td>
</tr>
<tr>
<td>DCL NO.</td>
<td>IDENTIFIER</td>
<td>ATTRIBUTES AND REFERENCES</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>187</td>
<td>I</td>
<td>AUTOMATIC, ALIGNED, BINARY, FIXED(15,2) 36, 30, 39, 43, 47, 48, 49, 50, 52, 53, 72, 75, 77, 92, 93, 96, 99, 121, 122, 123, 127, 128, 139, 141, 142, 146, 148, 149, 152, 151</td>
</tr>
<tr>
<td>188</td>
<td>INITIAL_MESSAGE</td>
<td>AUTOMATIC, UNALIGNED, STRING(41), CHARACTER</td>
</tr>
<tr>
<td>23</td>
<td>INITIALIZE_SCREEN</td>
<td>STATEMENT LABEL CONSTANT</td>
</tr>
<tr>
<td>187</td>
<td>J</td>
<td>AUTOMATIC, ALIGNED, BINARY, FIXED(15,0) 62, 63, 64, 66, 68</td>
</tr>
<tr>
<td>7</td>
<td>LETTER</td>
<td>AUTOMATIC, UNALIGNED, STRING(50), CHARACTER, VARYING 4, 13</td>
</tr>
<tr>
<td>7</td>
<td>LETTER_OPTION</td>
<td>AUTOMATIC, ALIGNED, BINARY, FIXED(15,0) 4, 13</td>
</tr>
<tr>
<td>188</td>
<td>LINE</td>
<td>101111 AUTOMATIC, UNALIGNED, INITIAL, STRING(41), CHARACTER 27, 75, 77, 79, 99, 112, 113, 115, 117</td>
</tr>
<tr>
<td>188</td>
<td>LNE</td>
<td>AUTOMATIC, ALIGNED, INITIAL, BINARY, FIXED(15,0) 76, 78, 80, 99, 113, 115, 117</td>
</tr>
<tr>
<td>7</td>
<td>LOGICAL_OPERATOR</td>
<td>AUTOMATIC, UNALIGNED, STRING(50), CHARACTER, VARYING 20</td>
</tr>
<tr>
<td>7</td>
<td>LOGICAL_OPERATOR_OPTION</td>
<td>AUTOMATIC, ALIGNED, BINARY, FIXED(15,0) 20</td>
</tr>
<tr>
<td>188</td>
<td>MESSAGE</td>
<td>AUTOMATIC, UNALIGNED, STRING(41), CHARACTER 27, 28, 32, 44, 46, 65, 66, 67, 73, 75, 76, 77, 78, 79, 81, 84, 87, 104, 112, 113, 114, 116, 124, 130, 134, 153, 160, 172, 175, 176, 186</td>
</tr>
<tr>
<td>7</td>
<td>NUL</td>
<td>STATIC, EXTERNAL, UNALIGNED, STRING(0), CHARACTER 2, 6</td>
</tr>
<tr>
<td>7</td>
<td>NUMBER</td>
<td>AUTOMATIC, UNALIGNED, STRING(50), CHARACTER, VARYING 12</td>
</tr>
</tbody>
</table>
```plaintext
DCL NO.    IDENTIFIER                ATTRIBUTES AND REFERENCES
7         NUMBER OPTION             AUTOMATIC, ALIGNED, BINARY, FIXED(15, 3), 12
108       ONSCOPE                  STATIC, EXTERNAL, ALIGNED, INITIAL, BINARY, FIXED(15, 6)
22        OPEN_SCOPE               STATEMENT LABEL CONSTANT 163
7         OPERATOR                 AUTOMATIC, UNALIGNED, STRING(50), CHARACTER, VARYING 16
7         OPERATOR OPTION          AUTOMATIC, ALIGNED, BINARY, FIXED(15, 0) 16
PARSE      EXTERNAL, ENTRY, DECIMAL, FLOAT(SINGLE) 30, 34, 57, 137, 167, 181
108       PARSE_RESULT             (50) AUTOMATIC, UNALIGNED, STRING(16C), CHARACTER, VARYING 48, 50, 68, 77, 115, 149, 151
108       PARSE_RESULT OPTION      (50) AUTOMATIC, ALIGNED, BINARY, FIXED(15, 0) 49, 50, 64, 66, 72, 75, 111, 113, 150, 151
108       PARSE_RULE               (50) AUTOMATIC, UNALIGNED, STRING(16C), CHARACTER, VARYING 39, 47, 63, 99, 100, 110, 120, 121, 127, 128, 135, 142, 148, 151, 159, 167, 169, 181
108       PARSE_RULE_INDEX          AUTOMATIC, ALIGNED, INITIAL, BINARY, FIXEDU(15, 0) 38, 42, 42, 43, 62, 90, 98, 119, 126, 135, 141, 145, 146, 166, 171, 186
108       PARSE_RULE_NAME OPTION   AUTOMATIC, ALIGNED, BINARY, FIXED(15, 0) 6, 182
108       PARSE_RULE_POSITION      AUTOMATIC, ALIGNED, INITIAL, BINARY, FIXED(15, 0) 93, 96, 97, 98, 99, 100, 102, 102
175       PROCESSING COMPLETE      STATEMENT LABEL CONSTANT 33, 51, 85, 88, 105, 117, 125, 131, 133, 154, 161, 173
187       PUNCH                     FILE, EXTERNAL, PRINT 128
26        REAC REQUEST             STATEMENT LABEL CONSTANT
```
<table>
<thead>
<tr>
<th>DCL NO.</th>
<th>IDENTIFIER</th>
<th>ATTRIBUTES AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>RELATIONAL_OPERATOR</td>
<td>AUTOMATIC, UNALIGNED, STRING(50), CHARACTER, VARYING 19</td>
</tr>
<tr>
<td>7</td>
<td>RELATIONAL_OPERATOR_OPTION</td>
<td>AUTOMATIC, ALIGNED, BINARY, FIXED(15, 0)</td>
</tr>
<tr>
<td>108</td>
<td>REMAINING_INPUT</td>
<td>AUTOMATIC, UNALIGNED, STRING(160), CHARACTER, VARYING 30, 34, 47, 50, 57, 136, 137, 146, 167</td>
</tr>
<tr>
<td>107</td>
<td>REMAINING_INPUT2</td>
<td>AUTOMATIC, UNALIGNED, STRING(80), CHARACTER, VARYING 34, 57, 79, 137, 179, 181</td>
</tr>
<tr>
<td>188</td>
<td>REMAINING_INPUT3</td>
<td>AUTOMATIC, UNALIGNED, STRING(80), CHARACTER, VARYING 101</td>
</tr>
<tr>
<td>188</td>
<td>REQUEST</td>
<td>AUTOMATIC, UNALIGNED, STRING(160), CHARACTER, VARYING 26, 27, 33</td>
</tr>
<tr>
<td>187</td>
<td>ROUTINE</td>
<td>(0:15) AUTOMATIC, INITIAL, LABEL 32, 34, 52, 80, 89, 106, 119, 126, 132, 155, 162, 166, 174, 31</td>
</tr>
<tr>
<td>188</td>
<td>RSIB</td>
<td>AUTOMATIC, ALIGNED, INITIAL, BINARY, FIXED(15, 0)</td>
</tr>
<tr>
<td>84</td>
<td>RULE DOES NOT_EXIST</td>
<td>STATEMENT LABEL CONSTANT 94, 106, 157</td>
</tr>
<tr>
<td>187</td>
<td>SGHMESS</td>
<td>EXTERNAL, ENTRY, DECIMAL, FLOAT(SINGLE) 164</td>
</tr>
<tr>
<td>7</td>
<td>SIMPLE_VARIABLE</td>
<td>AUTOMATIC, UNALIGNED, STRING(50), CHARACTER, VARYING 3, 10, 29, 46, 57, 73, 81, 84, 89, 112, 113, 116, 158, 165, 168, 179</td>
</tr>
<tr>
<td>7</td>
<td>SIMPLE_VARIABLE_OPTION</td>
<td>AUTOMATIC, ALIGNED, BINARY, FIXED(15, 0)</td>
</tr>
<tr>
<td>188</td>
<td>SMI</td>
<td>AUTOMATIC, ALIGNED, INITIAL, BINARY, FIXED(15, 0)</td>
</tr>
<tr>
<td>7</td>
<td>SPECIFIED</td>
<td>AUTOMATIC, ALIGNED, INITIAL, BINARY, FIXED(15, 0)</td>
</tr>
<tr>
<td>188</td>
<td>START_SYMBOL</td>
<td>AUTOMATIC, UNALIGNED, STRING(11), CHARACTER 21, 176</td>
</tr>
</tbody>
</table>
TEST_SYNTAX: PROCEDURE OPTIONS (MAIN); /* FILE TESTSYN */

<table>
<thead>
<tr>
<th>DCL NO.</th>
<th>IDENTIFIER</th>
<th>ATTRIBUTES AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>SUBSCRIPT</td>
<td>AUTOMATIC, UNALIGNED, STRING(50), CHARACTER, VARYING 9</td>
</tr>
<tr>
<td>7</td>
<td>SUBSCRIPT_OPTION</td>
<td>AUTOMATIC, ALIGNED, BINARY, FIXED(15,0)</td>
</tr>
<tr>
<td>7</td>
<td>SUCCESSFULLY</td>
<td>AUTOMATIC, ALIGNED, INITIAL, BINARY, FIXED(15,0)</td>
</tr>
<tr>
<td></td>
<td>SYSIN</td>
<td>FILE, EXTERNAL</td>
</tr>
<tr>
<td>1</td>
<td>TEST_SYNTAX</td>
<td>ENTRY, DECIMAL, FLOAT(SINGLE)</td>
</tr>
<tr>
<td>187</td>
<td>TRACE</td>
<td>AUTOMATIC, ALIGNED, INITIAL, BINARY, FIXED(15,0)</td>
</tr>
<tr>
<td></td>
<td>TRACE_PARSE</td>
<td>EXTERNAL, ENTRY, DECIMAL, FLOAT(SINGLE)</td>
</tr>
<tr>
<td>7</td>
<td>UNARY_OPERATOR</td>
<td>AUTOMATIC, UNALIGNED, STRING(50), CHARACTER, VARYING 17</td>
</tr>
<tr>
<td>7</td>
<td>UNARY_OPERATOR_OPTION</td>
<td>AUTOMATIC, ALIGNED, BINARY, FIXED(15,0)</td>
</tr>
<tr>
<td>188</td>
<td>UNIT</td>
<td>AUTOMATIC, ALIGNED, INITIAL, BINARY, FIXED(15,0)</td>
</tr>
<tr>
<td></td>
<td>UNSPEC</td>
<td>GENERIC, BUILT-IN FUNCTION</td>
</tr>
<tr>
<td>7</td>
<td>UNSPECIFIED</td>
<td>AUTOMATIC, ALIGNED, INITIAL, BINARY, FIXED(15,0)</td>
</tr>
<tr>
<td>7</td>
<td>UNSUCCESSFULLY</td>
<td>AUTOMATIC, ALIGNED, INITIAL, BINARY, FIXED(15,0)</td>
</tr>
<tr>
<td>7</td>
<td>VARIABLE</td>
<td>AUTOMATIC, UNALIGNED, STRING(50), CHARACTER, VARYING 8</td>
</tr>
<tr>
<td>7</td>
<td>VARIABLE_OPTION</td>
<td>AUTOMATIC, ALIGNED, BINARY, FIXED(15,0)</td>
</tr>
<tr>
<td>DCL NO.</td>
<td>IDENTIFIER</td>
<td>ATTRIBUTES AND REFERENCES</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>24</td>
<td>WAIT</td>
<td>STATEMENT LABEL CONSTANT 177</td>
</tr>
</tbody>
</table>
TEST_Syntax: PROCEDURE OPTIONS(IN MAIN); /* FILE TESTSYN */

<table>
<thead>
<tr>
<th>STATEMENT NO.</th>
<th>IDENTIFIER</th>
<th>LENGTH IN BYTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>188</td>
<td>LINE</td>
<td>12</td>
</tr>
<tr>
<td>187</td>
<td>PARSE_RESULT</td>
<td>500G</td>
</tr>
<tr>
<td>187</td>
<td>PARSE_RESULT_OPTION</td>
<td>200</td>
</tr>
<tr>
<td>187</td>
<td>PARSE_RULE</td>
<td>800G</td>
</tr>
<tr>
<td>187</td>
<td>ROUTINE</td>
<td>128</td>
</tr>
</tbody>
</table>
TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSY */

STORAGE REQUIREMENTS.

THE STORAGE AREA FOR THE PROCEDURE LABELLED TEST_SYNTAX IS 16B20 BYTES LONG.
THE STORAGE AREA FOR THE UNIT AT STATEMENT NO. 132 IS 144 BYTES LONG.
THE STORAGE AREA (IN STATIC) FOR THE PROCEDURE LABELLED FIND_NAME IS 352 BYTES LONG.
THE PROGRAM CSECTION IS NAMED TESTTAX AND IS 9010 BYTES LONG.
THE STATIC CSECTION IS NAMED TESTTAXA AND IS 4264 BYTES LONG.
TESTSyntax: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */

COMPILER DIAGNOSTICS.

ERRORS.

IEM2867I IMPLEMENTATION RESTRICTION. EXTERNAL NAME TESTSyntax HAS BEEN TRUNCATED TO 7 CHARACTERS.

IEM2867I IMPLEMENTATION RESTRICTION. EXTERNAL NAME DEFINEPARSE_REQUEST HAS BEEN TRUNCATED TO 7 CHARACTERS.

IEM2867I IMPLEMENTATION RESTRICTION. EXTERNAL NAME COMPILEPARSE_REQUEST HAS BEEN TRUNCATED TO 7 CHARACTERS.

IEM2867I IMPLEMENTATION RESTRICTION. EXTERNAL NAME TRACENPARSE HAS BEEN TRUNCATED TO 7 CHARACTERS.

IEM2867I IMPLEMENTATION RESTRICTION. EXTERNAL NAME DELETENPARSE_REQUEST HAS BEEN TRUNCATED TO 7 CHARACTERS.

WARNINGS.

IEM0227I NO FILE/STRING OPTION SPECIFIED IN ONE OR MORE GET/PUT STATEMENTS. SYSIN/SYSPRINT HAS BEEN ASSUMED IN EACH CASE.

END OF DIAGNOSTICS.
Appendix B

BNF DEFINITION OF APAREL'S SYNTAX LANGUAGE

\[
\text{(PARSE_REQUEST)} := (\text{PARSE_DELIMITER}) (\text{PARSE_NAME}) : \text{(PARSE_ALTERNATIVE LIST)} (\text{PARSE_DELIMITER}) \\
\text{(PARSE_ALTERNATIVE LIST)} := (\text{PARSE_ALTERNATIVE NAME}) \\
\text{(PARSE_ELEMENT_LIST)} | (\text{PARSE_ALTERNATIVE_NAME}) \\
\text{(PARSE_ELEMENT_LIST)} | \text{(PARSE_ALTERNATIVE_LIST)} \\
\text{(PARSE_ELEMENT_LIST)} := (\text{PARSE_ELEMENT}) | (\text{PARSE_ELEMENT}; (\text{PARSE_TIME_ROUTINE_NAME}) | (\text{PARSE_ELEMENT}) (\text{PARSE_ELEMENT_LIST}) | (\text{PARSE_ELEMENT}). (\text{PARSE_ELEMENT_LIST}) \\
\text{(PARSE_ELEMENT)} := (\text{PARSE_ATOM}) | (\text{PARSE_GROUP}) \\
\text{(PARSE_GROUP)} := (\text{PARSE_ALTERNATIVE_NAME}) | (\text{PARSE_NAME}) (\text{PARSE_ALTERNATIVE_LIST}) | (\text{PARSE_NAME}) : (\text{PARSE_ALTERNATIVE_LIST}) \\
\text{(PARSE_ATOM)} := (\text{PARSE_NAME}) | (\text{TEXT_LITERAL}) \\
\text{(PARSE_NAME)} := (\text{PL/1 IDENTIFIER}) \\
\text{(PARSE_ALTERNATIVE_NAME)} := (\text{PARSE_NAME}) \\
\text{(PARSE_DELIMITER)} := :: \\
\text{(PARSE_TIME_ROUTINE_NAME)} := (\text{NAME OF A PL/1 BIT VALUED FUNCTION})
\]
REFERENCES


2. IBM System/360, Principles of Operation, Form A22-6821, IBM Corporation.