Burroughs

PUBLICATION CHANGE NOTICE

PCN No.: 5001530-001 Date: March, 1981

Publication Title: B 6000/B 7000 PL/I Language Reference Manual

Other Affected Publications: None

Supersedes: N/A

Description:

Change Pages		Add Pages
TOC-i	7-45	7-21B
TOC-iii	7-53	7-46A
TOC-v	8-1	7-46C
TOC-vii	8-23	
TOC-ix	A1-11	
TOC-xi	A2-1	
7-21	A2-9	
7-33	A7-29	
7-35	A7-33	
7-37		
7-39		
7-41		
7-43		

Retain this PCN as a record of changes made to the basic publication.

The above pages covering
PCN 5001530-001

Burroughs believes that the information described in this manual is accurate and reliable, and much care has been taken in its preparation. However, no responsibility, financial or otherwise, is accepted for any consequences arising out of the use of this material. The information contained herein is subject to change. Revisions may be issued to advise of such changes and/or additions.

Correspondence regarding this document should be addressed directly to Burroughs Corporation, P.O. Box 4040, El Monte, California 91734, Attn: Publications Department, TIO–West.

TABLE OF CONTENTS

Syntax

```
<stop-statement> ::= STOP;
```

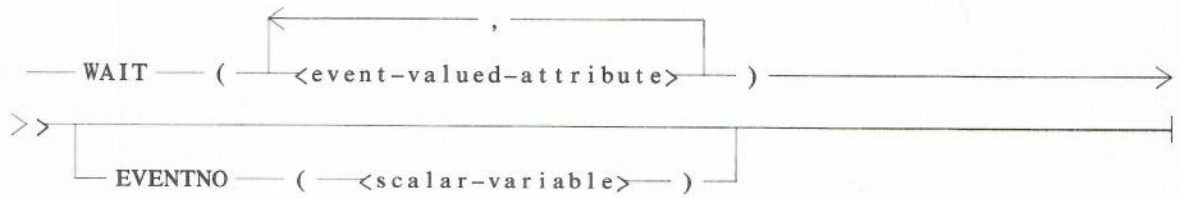
Semantics

Prior to any termination activity the FINISH condition is raised in the task in which the STOP is executed. On normal return from the finish on-unit, all tasks in the program are terminated.

7.3.2.10 The WAIT Statement

The WAIT statement causes suspension of a task until one of the listed <event-valued-attribute>s is caused.

Syntax



Semantics

If the EVENTNO clause is used, the ordinal position of the <event-valued-attribute> that caused activation of the task is stored in <scalar-variable>. <scalar-variable> may be a pseudovariable.

7.3.3 Program Structure Statements

7.3.3.1 The BEGIN Statement

The BEGIN statement is the heading statement of a begin block.

Syntax

```
<begin-statement> ::=BEGIN [OPTIONS
                          (<begin-option-list>)];

<begin-option-list> ::= DOUBLE
```

Semantics

A BEGIN statement is used in conjunction with an END statement.

See Section 6.3, Blocks for a discussion of blocks.

Examples

```
1.  ON OVERFLOW BEGIN;
                  .
                  .
                  .
                END;
2.  (SIZE):  Q: PROCEDURE;
                  .
                  .
                  .
      (NOSIZE): A: BEGIN
                  .
                  .
                  .
                END;
                  .
                  .
                  .
              END;
```

The SIZE condition is enabled with the prefix to the PROCEDURE statement. This enabling is negated throughout the begin block with the prefix NOSIZE. On exit from the begin block, SIZE errors are again enabled because statements again are in the scope of the SIZE prefix.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

7.3.3.2 The END Statement

The END statement terminates blocks and do-groups.

Syntax

```
<end-statement> ::= END [<label>];
```

Semantics

If a label follows the END, the END statement terminates the block or do-group that is headed by the nearest preceding heading statement having that label. It also terminates all unclosed blocks and do-groups that are physically within that block or group.

If a label does not follow the END, the END statement terminates that group or block headed by the nearest preceding DO, BEGIN, or PROCEDURE statement for which there is no other corresponding END statement.

If control reaches an END statement terminating a procedure, it is treated as a RETURN statement.

If control reaches an END statement which terminates a begin block that is an on-unit, control is returned to the point specified for that particular interrupt.

If a label follows the END, that label may not be an element of a label array.

7.3.3.3 The ENTRY Statement

The ENTRY statement specifies a secondary entry point to a procedure.

Syntax

```
<entry-statement> ::=<entry-name>: ... ENTRY
    [(<parameter>[,<parameter>]...)]

     [RETURNS(<data-specification-attributes>)];
```

Semantics

The parameters are names that specify the parameters of the entry point.

The <data-specification-attributes> permitted for the RETURNS attribute of an ENTRY statement are the ARITHMETIC, STRING, AREA, OFFSET, ENTRY, and POINTER attributes. These attributes specify the characteristics of the value returned by the procedure when invoked as a function by any of the entry names. The value specified in the RETURN statement of the invoked entry is converted, if necessary, to conform to the specified attributes.

If an ENTRY statement has more than one label, each label is interpreted as if it were a single entry name for a separate ENTRY statement having the same parameter list. If <data-specification-attributes> are specified, they apply to all

7.3.5.4 The SIGNAL Statement

The SIGNAL statement simulates the occurrence of the named condition and causes an interrupt if the condition is enabled. It may be used to test the action specification of the current ON statement. The result is that the <on-unit> will be executed.

Syntax

```
<signal-statement> ::= SIGNAL <condition>;
```

Examples

```
1.     X:  PROCEDURE;
             .
             .
             .
      ON1:  ON ENDFILE (DATIN) Y,Z = 0;
             .
             .
             .
       S1:  SIGNAL ENDFILE (DATIN);
             .
             .
             .
      ON2:  ON ENDFILE (DATIN) SYSTEM;
             .
             .
             .
       S2:  SIGNAL ENDFILE (DATIN);
             .
             .
             .
           END X;
```

Statement S1 causes an interrupt in the same way as if an attempt to read past a file delimiter had actually occurred. Control is transferred to the statement Y,Z = 0; in the ON1 statement.

When statement S2 causes an interrupt, control is transferred to the ON2 statement, and standard system action is taken.

```
2.   ON CONDITION (TAX) TAXCT = TAXCT+1;
       .
       .
       .
     SIGNAL CONDITION (TAX);
```

The ON statement establishes an action for the programmer-specified condition TAX. This condition can occur only when a SIGNAL statement causes the condition to occur.

7.3.6 Input/Output Statements

7.3.6.1 The CLOSE Statement

The CLOSE statement makes the named file inaccessible in the current task.

Syntax

```
<close-statement> ::=
    CLOSE<options-group> [,<options-group>]...;

<options-group> ::=
    FILE(<file-name>)
   [SUBFILE(<scalar-expression>)]
    [VOLUME | {{ENVIRONMENT | OPTIONS} (<close-option>)}]


<close-option> ::= PURGE | LOCK | CRUNCH | NOREWIND
```

Semantics

The options may appear in any order within an <options-group>.

The FILE option specifies the file to be closed. Several files may be closed by one CLOSE statement via multiple <options-group>s forming a multiple CLOSE statement.

Closing an unopened file, or a closed file, has no effect.

The SUBFILE clause has no effect on a CLOSE statement for non-PORT files. For PORT files, the value of <scalar-expression> in the SUBFILE clause determines which subfile is to be closed. If this value is zero, all open subfiles are closed. If this value is greater than zero but less than or equal to the value of MAXSUBFILES, only the specified subfile is closed. A bad subfile index is ignored.

Because only one meaning exists for CLOSE on a PORT file, any <close-option>s specified in the CLOSE statement are ignored.

All I/O event variables associated with operations on the file that have not been completed before the file is closed are set complete, with a status value of 1 if not already non-zero.

When no VOLUME, ENVIRONMENT, or OPTIONS option appears, an end-of-file mark, record, or label is placed in the file and the file is closed. A temporary disk file is removed from the directory, and a tape file is rewound.

When the VOLUME option appears, the file must be a tape file. On multi-file tape reels, the tape is positioned after file closure at the next file on the reel.

When the ENVIRONMENT(PURGE) or OPTIONS (PURGE) option appears, the file is closed, purged, and released to the system. A disk file is removed from the directory.

When the ENVIRONMENT(LOCK) or OPTIONS (LOCK) option appears, the file is closed, locked, and saved by the system. A tape file is rewound.

When the ENVIRONMENT (CRUNCH) or OPTIONS (CRUNCH) option appears , the file must be a disk file. The file is closed, locked, and the unused portion of the last row of disk space (beyond the end-of-file indication) is returned to the system. For KEYED disk files CRUNCH implies LOCK.

When the ENVIRONMENT (NOREWIND) or OPTIONS (NOREWIND) option appears, the file must be a tape file. The file is closed and the tape remains positioned at the beginning of the next file. This option is identical to the VOLUME option.

Examples

```
CLOSE FILE(MASTER);
CLOSE FILE(TABLEA)OPTIONS(LOCK),
      FILE(TABLEB)OPTIONS(PURGE);
CLOSE FILE(GN)VOLUME;
```

7.3.6.2 The DELETE Statement

The DELETE statement deletes a record from an UPDATE file.

Syntax

```
<delete-statement> ::= DELETE <option-list>;
<option-list> ::=
    FILE (<file-name>) [KEY (<scalar-expression>)]
```

Semantics

The options may appear in any order.

The FILE (<file-name>) option specifies an UPDATE file and must be specified.

The KEY option must be specified if the file is a DIRECT UPDATE file. It cannot be specified otherwise. The expression is converted to a character string and determines which record is to be deleted.

If the file is a SEQUENTIAL UPDATE file, the record to be deleted is the last record that was read.

The DELETE statement unlocks a record only if that record had been locked in the same task in which the DELETE appears.

The DELETE statement can cause implicit opening of a file. The system file attribute MYUSE is set equal to 'UPDATE' for an implicitly opened file.

Example

```
DELETE FILE(ALPHA) KEY (DKEY);
```

This statement causes the record identified by DKEY to be deleted from the data set associated with the file ALPHA. If the record was previously locked in the same task, it is unlocked.

7.3.6.3 The DISPLAY Statement

The display statement causes a message to be displayed at the supervisory console. A response may be requested.

Syntax

```
<display-statement> ::=

     DISPLAY (<scalar-expression>)
     [REPLY (<scalar-character-variable>)]
```

Semantics

In the following, the format of the DISPLAY statement without the option is referred to as option 1, and with the REPLY option as option 2.

Execution of the DISPLAY statement causes the <scalar-expression> to be evaluated and, where necessary, converted to a varying character string. This character string is the message to be displayed.

In option 2, the character variable receives a string that is a message to be supplied by the operator.

Example

```
DISPLAY ('END OF JOB');
```

This statement causes the message, "END OF JOB" to be displayed at the supervisory console.

7.3.6.4 The FORMAT Statement

The FORMAT statement specifies a format list for use with data transmitted under edit direction.

Syntax

```
<format-statement> ::=
    <label>: [<label>:] ... FORMAT (<format-list>);
```

Semantics

The <format-list> is as described for use with an EDIT-directed data specification. See Section 8.4.6.1, Format Lists.

At least one <label> is required.

A GET or PUT statement may include a remote format specification. See Section 8.4.6.3.4, Remote Format Item.

The remote format item and the FORMAT statement must be internal to the same block.

A FORMAT statement encountered in sequential flow of control is treated as a no-operation.

It is an error to attempt to transfer control to a FORMAT statement by means of a GO TO statement.

7.3.6.5 The GET Statement

The GET statement normally causes values from a data set to be assigned to variables specified in a data list. Alternatively, the values may come from a character-string variable.

Syntax

```
<get-statement> ::=GET <option-list>;

<option-list> ::=

        {[FILE(<file-name>) [COPY] [SKIP
        [(<scalar-expression>)]]] |

      STRING(<scalar-character-string-variable>)}
           [<data-specification>];
```

Semantics

If neither the FILE(<file-name>) option nor the STRING (<scalar-character-string-variable>) option appears, the file option FILE(SYSIN) is assumed. (Default <file-name> is SYSIN.)

The <data-specification> must appear unless the SKIP option is specified.

The options may appear in any order.

The <file-name> refers to the file which is to provide the values. It must be a STREAM INPUT file.

The <scalar-character-string-variable> refers to the character string that is to provide the data to be assigned to the data list. This name may be a reference to a character string built-in function. Each GET operation using this option always begins at the beginning of the specified string. If the number of characters in this string is less than the total number of characters implied by the data specification, the ERROR condition is raised.

When the STRING option is used under data-directed transmission, the ERROR condition is raised if an identifier within the string does not have a match within the data specification.

For the rules concerning data specification see Section 8.4.3, Data Lists.

If the FILE<file-name> option refers to a file that is not open, the file is implicitly opened for the stream input transmission; the system file attribute MYUSE is set equal to 'INPUT' for the file.

The COPY option, which may only be used with the FILE option, specifies that the source data, as read, is to be written, without alteration, on the standard installation print file.

The SKIP option, which may only be used with the FILE option, causes a new current line to be defined for the data set. The expression, if present, is converted to an integer W, which must be greater than zero. The data set is positioned at the start of the W-th line relative to the current line. If the expression is omitted, SKIP(1) is assumed. The SKIP option always is executed before any data is transmitted. If the first action on a stream input file is a GET SKIP statement, the data set is positioned at the start of the first line.

Examples

1. GET LIST (A,B,C);

Specifies the list directed transmission of the values to be assigned to A, B, and C from the file named SYSIN.

2. GET FILE (BETA) EDIT (X,Y,Z) (A(5), F(5,2), A(10));

Specifies the edit-directed transmission of the values assigned to X, Y, and Z from file BETA.

3.
```
C:PROC;
      ON ENDFILE(SYSIN) GOTO X;
      L:GET COPY SKIP;
      GOTO L;
X:END C;
```

Reads all cards in the default input file and prints them on the SYSPRINT output file.

7.3.6.6 The LOCATE Statement

The LOCATE statement, which applies to output files, causes allocation of the specified BASED variable in a buffer. It may also cause transmission of a BASED variable previously allocated in a buffer.

Syntax

<locate-statement> ::= LOCATE <variable> <option-list>;

<option-list> ::=

FILE (<file-name>)
[SET(<scalar-pointer-variable>)]
[KEYFROM(<scalar-expression>)]

Restrictions:

The <variable> must be an unsubscripted level 1 BASED variable.

Semantics

The options in the <option-list> may appear in any order.

The FILE(<file-name>) clause specifies the file involved. This clause must appear.

Execution of the LOCATE statement causes the specified based variable to be allocated in the buffer. Components of the based variable that have been given the INITIAL attribute, or components specified in REFER options, are initialized. A pointer value is assigned to the pointer variable named in the SET option or, if the SET option is omitted, to the pointer variable specified in the declaration of the BASED variable. The pointer value identifies the record in the buffer. If the pointer variable is an OFFSET variable, the pointer value is implicitly converted. After execution of the LOCATE statement, values may be assigned to the based variable for subsequent transmission to the file, which will occur immediately before the next LOCATE, WRITE, or CLOSE operation on the file, at which time the record is freed.

IF the KEYFROM (<scalar-expression>) option appears, the value of the <scalar-expression> is converted to a character string and is used as the key of the record when it is subsequently written.

If the FILE(<file-name>) clause refers to a file that is not open, the file is implicitly opened; the system file attribute MYUSE is set equal to 'OUTPUT' for the file.

Example

```
LOCATE ALPHA SET (REC-POINT) FILE (BETA);
```

The BASED variable ALPHA is allocated in a buffer and the <scalar-pointer-variable> REC-POINT is set to identify ALPHA in the buffer. Values may subsequently be assigned to ALPHA and the record will be written in the data set associated with the file BETA when a subsequent LOCATE or WRITE statement is executed for file BETA or if BETA is closed, either explicitly or implicitly.

7.3.6.7 The OPEN Statement

The OPEN statement completes the specification of attributes for a file and explicitly opens a file if it has not been previously opened. If the file has already been opened, the statement is ignored. Attribute specifications in the OPEN statement take precedence over attribute specifications in the file declaration.

Syntax

```
<open-statement> ::= OPEN <options-group>
                     [,<options-group>]...;

<options-group> ::=
    FILE(<file designator>)
    [TITLE (<file-title>)]
    [{ENVIRONMENT | OPTIONS}
    (<system-file-attribute-specification-list>)]
    [PRINT]
    [INPUT | OUTPUT | UPDATE ]
    [STREAM | RECORD]
    [LINESIZE(<scalar-expression>)]
    [PAGESIZE(<scalar-expression>)]
    [SUBFILE(<scalar-expression>)]
    [SEQUENTIAL | DIRECT]
    [TAB(<scalar-expression list>)]
    [WAIT | OFFER | AVAILABLE]

<file-title> ::= <character-string>

<system-file-attribute-specification-list>::=
    <system-file-attribute-specification>
    [,<system-file-attribute-specification>]...

<system-file-attribute-specification> ::=
    <system-file-attribute> = <scalar-expression> |
    <Boolean-valued-system-file-attribute>
```

Semantics

The INPUT, OUTPUT, UPDATE, STREAM, RECORD, DIRECT, SEQUENTIAL, KEYED, PRINT, and ENVIRONMENT (or OPTIONS) options specify attributes which may augment or override the attributes specified in the file declaration.

The options may appear in any order within a group.

The FILE (<file-name>) clause specifies which file is to be opened. The clause must appear once in each options group. An OPEN statement can be used to open several files, by multiple use of the options-group. This is known as a multiple statement, and is equivalent to a list of single OPEN statements, where the options groups in the list are taken in left-to-right order from the multiple OPEN statement.

If a file has been opened in a task and not subsequently closed, then reopening this file in the same task or a descendant task has no effect on the file. All options (including TITLE) are evaluated whether or not they conflict with the options of the previous open, but they are not used. If a file has been opened and subsequently closed, it may be reopened.

The TITLE option may be used to specify the name by which the system is to identify the file. The default TITLE is the <file-name> in the FILE option. In the case of a parameter, the corresponding argument identifier is the default title.

The ENVIRONMENT or OPTIONS option is identical in form to the ENVIRONMENT attribute as discussed in Section 4.8.7, Environment Attribute.

The LINESIZE option can be specified only for a STREAM OUTPUT file. The expression is evaluated, converted to an integer, and used as the length of a line during subsequent operations on the file. New lines may be started by use of the printing and control format items or by options in a GET or PUT statement. If an attempt is made to position a file past the end of a line before explicit action to start a new line is taken, a new line is automatically started, and the file is positioned to the start of this new line.

The LINESIZE option cannot be specified for an INPUT file. The linesize taken into consideration whenever a SKIP option appears in a GET statement is the linesize that was used to create the file.

The PAGESIZE option can be specified only for a STREAM PRINT file. The expression is converted to an integer and used as the number of lines on a page. During subsequent output to the file, new pages may be started by use of the PAGE format item or PUT PAGE statement option. If a page overflows before action to start a new page is given, the ENDPAGE condition is raised.

The SUBFILE option has no effect on OPEN statements for non-PORT files. For PORT files, the value of <scalar-expression> in the SUBFILE clause specifies which subfile is to be opened. If this value is zero, the OPEN statement opens all subfiles.

Attempting to set conflicting language file attributes will generate a syntax error of level three. For a list of compatible language file attributes, refer to Section 8.1, File Attributes.

Attempting to set a system file attribute inside the environment or options clause that conflicts with a language attribute will be ignored.

Syntax for setting system file attributes inside the OPTIONS or ENVIRONMENT clause in an OPEN statement is the same as for a file declaration except that the specifications need not be constant. In the case of system file attributes that require mnemonics, they must be used. This can be done by specifying a string constant or a string variable which has been assigned the correct value. If a constant is used, a compile time check is made to see if it is a valid mnemonic for the specified file attribute. If not, an error of level three is given. If a variable is used, it is checked at run time, and if it is an invalid mnemonic, a non-fatal run time error is given.

Attempting to set a read-only attribute inside the ENVIRONMENT or OPTIONS clause will cause a syntax error error of level three to be generated and the specification will be ignored.

When a TAB (<scalar-expression-list>) clause appears the <scalar-expression> list defines the tab columns to be used for list and data directed PUT statements or for tab format items. The expressions must be ascending, positive and less than the file line size. All erroneous expressions will be ignored.

The WAIT, AVAILABLE, and OFFER options apply only to PORT files. WAIT is the default value; the subfile or subfiles are offered for matching, and the program is suspended, until a matching subfile is found for all offered subfiles. OFFER causes the subfile or subfiles to be offered for matching; the program is resumed without waiting for the subfiles to be matched. AVAILABLE causes the subfiles to be matched only to complementary subfiles that are already offered. If a match is found for a particular subfile, that subfile is opened; if no match is found, a NOFILEFOUND result is returned for that subfile, and the subfile is not left offered for subsequent matching.

Examples

1. OPEN FILE (ALPHA), FILE (BETA) TITLE ('WORKFILE');

   The files ALPHA and BETA are opened. The data set associated with BETA is identified through use of the external name WORKFILE, whereas ALPHA is identified with a data set through use of the external name ALPHA.

2. OPEN FILE (MASTER) UPDATE;

   The file MASTER is opened as an UPDATE file. MASTER is by default the name used to associate a data set with the file.

3. OPEN FILE (SYSPRINT) LINESIZE (132) TAB (20,40,105);

   If the TAB settings are not supplied, the following default TABS will be used (1, 25, 49, 73, 47, 121, ...).

7.3.6.8 The PUT Statement

The PUT statement causes the transmission of data and/or the execution of control options. Data items transmitted are the character string representations of values of expressions that are assigned to a data set or to a designated character string variable.

Syntax

```
<put-statement> ::= PUT <option-list>;

<option-list> ::=
   {[FILE(<file-name>)] | STRING
   (<scalar-character-string-variable>)}
         [<data-specification>] [PAGE]
         [SKIP [(<scalar-expression>)]]
         [LINE (<scalar-expression>)]
```

Semantics

The PAGE, SKIP, and LINE options cannot be used with the STRING option.

If neither the FILE(<file-name>) option nor the STRING (<scalar-character-string-variable>) option appears, the file option FILE(SYSPRINT) is assumed.

The <file-name> refers to a file that is to receive the values. It must be a STREAM OUTPUT file.

The <scalar-character-string-variable> refers to the character string variable or pseudo-variable that is to receive the values. After appropriate conversion, the data specified by the <data-specification> is assigned to the string starting at the leftmost character (leftmost specified character in the case of a SUBSTR pseudo-variable). Any subsequent PUT statement will cause assignment to begin at the same place. If the string is not long enough to accommodate the data, the ERROR condition is raised.

The options may appear in any order. The PAGE and LINE options can be specified for PRINT files only. All of the options take effect before transmission of any values defined by the <data-specification>, if given. Of the three options PAGE, SKIP, and LINE, only PAGE and LINE may appear in the same PUT statement, in which case, the PAGE option is applied first.

The PAGE option causes a new current page to be defined within the file. If a data specification is present, the transmission of values occurs after the definition of the new page. The page remains current until the execution of a PUT statement with the PAGE option, until a PAGE format item is encountered, or until an ENDPAGE interrupt results in the definition of a new page. A new current page implies line one.

The SKIP option causes a new current line to be defined for the file. The expression, if present, is converted to an integer W, which for non-print files must be greater than zero. The file is positioned at the start of the W-th line relative to the current line. If the expression is omitted, SKIP(1) is assumed.

For PRINT files, W may be less than or equal to zero. In this case, the effect is that of a carriage return with the same current line. If less than W lines remain on the current page when a SKIP(W) is issued, ENDPAGE is raised.

The LINE option causes a current line to be defined for the file. The expression is converted to an integer W. If W specifies the current line of the most recent PUT statement, no new current line is established. If W is greater than the current line, blank lines are inserted so that the next line will be the W-th line of the current page. If more than W lines have already been written on the current page or if W exceeds PAGESIZE of the file, the ENDPAGE condition is raised. If W is less than or equal to zero, it is assumed to be one.

If the FILE(<file-name>) option refers to a file that is not open, the file is opened implicitly for stream output; the system file attribute MYUSE is set equal to 'OUTPUT' for the file.

Examples

1. PUT DATA (A,B,C);

   Specifies the data-directed transmission of the values A, B, and C to the file SYSPRINT.

2. PUT FILE (LIST) EDIT (X,Y,Z) (A(10)) PAGE;

   Specifies that a new page is to be defined for the print file list. The values of X, Y, and Z are placed starting in the first print position of the new page. Each of the values will use the A(10) format item.

7.3.6.9 The READ Statement

The READ statement causes a record to be transmitted from a RECORD INPUT or RECORD UPDATE file to a variable or buffer.

Syntax

```
<read-statement> ::= READ <option-list>;

<option-list> ::=
     FILE (<file-name>)
     { [INTO (<variable>)]|
       [SET (<scalar-pointer-variable>)]|
       [IGNORE (<scalar-expression>)]}

       [INDEX(<scalar-expression>)]

     { [KEY (<scalar-expression>)]|
       [KEYTO (<scalar-character-string-variable)] }

       [SUBFILE (<scalar-expression>)]


       [DONTWAIT (<scalar-expression>)]
```

Semantics

The options may appear in any order.

The FILE (<file-name>) clause specifies the file from which the record is to be read. This clause must appear. If the file specified is not open, it is implicitly opened; if the system file attribute MYUSE is not already 'UPDATE', it is changed to 'INPUT'.

The INTO(<variable>) option specifies an unsubscripted level 1 variable into which the record is to be read. It cannot be a parameter, nor can it have the DEFINED attribute.

The INDEX option may be used to specify a particular record of a tape or disk file to be read.

The KEY and KEYTO options can be specified for KEYED files only.

The KEY(<scalar-expression>) option must appear if the file is DIRECT. The expression is converted to a character string that determines which record is to be read.

The KEYTO(<scalar-character-string-variable>) option may be given only if the file is SEQUENTIAL KEYED. It specifies that the key of the record is to be copied into the string variable, which may be a pseudo-variable. This copying follows the rules for character string assignment, and if proper assignment cannot be made the KEY condition is raised. The key is the same key that was specified in the KEYFROM option when the record was written. KEYTO and KEY may not appear in the same READ statement.

The SET option specifies that the record is to be read into a buffer and that a pointer value is to be assigned to the named pointer variable. The pointer value identifies the record in the buffer.

The IGNORE option may be specified for SEQUENTIAL INPUT and SEQUENTIAL UPDATE files. The expression in the IGNORE option is evaluated and converted to an integer value. If this value, N, is greater than zero, N records are ignored. A subsequent READ statement for the file will access the (N+1)th record. If N <=0, no records are ignored. A READ statement without an INTO, SET, or IGNORE option is equivalent to a READ with an "IGNORE(1)".

A file with the KEYED attribute being accessed sequentially may be positioned by issuing a READ statement with the KEY option. The specified key will be used to identify the record required. Thereafter, records may be read sequentially from that point by use of READ statements without the KEY option. This applies to INPUT and UPDATE files.

For SEQUENTIAL files, two positioning statements can be used, with the following formats:

```
READ FILE (<file-name>)
    INTO (<variable>) KEY (<expression>);

and

READ FILE ( file name )
SET ( pointer-variable ) KEY ( expression)
```

The SUBFILE and NOWAIT clauses may be specified only for PORT files.

The value of <scalar-expression> in the SUBFILE clause determines which subfile in a PORT file is to be read. If this value is zero, a READ statement may read from any subfile. If <scalar-expression> is a variable or pseudovariable, after execution of the READ statement, the subfile number of the origin of the record that was read is stored in the variable or pseudovariable.

The value of <scalar-expression> in the DONTWAIT clause is converted to type BIT(1). If the resulting value is '0'B (or if the DONTWAIT clause is not specified), program execution waits until the read is finished. If the resulting value is '1'B, a read is done only if data are ready to be read. If <scalar-expression> is a variable or pseudovariable:

1. '1'B is stored in the variable or pseudovariable if the read was not done because no data were ready to be read.

2. '0'B is stored in the variable or pseudovariable if the read was done.

Execution of a READ statement with a SUBFILE clause or a DONTWAIT clause for non-PORT files results in the UNDEFINEDFILE condition.

For SEQUENTIAL files, only the first form shown immediately above can by used.

Examples

1. READ FILE (ALPHA) SET (REC-IDENT);

   The next record from the file ALPHA is made available and the pointer variable REC-IDENT is set to identify the record in the buffer.

2. READ FILE (BETA) KEY (VALUE) INTO (WORK);

   The record identified by the key value is transmitted from the file BETA into the variable WORK.

7.3.6.10 The REWRITE Statement

The REWRITE statement causes replacement of an existing record in a data set referred to be an UPDATE file.

Syntax

```
<rewrite-statement> ::= REWRITE <option-list>;
<option-list> ::=
     FILE (<file-name>) [KEY (<scalar-expression>)]
        [FROM (<variable>)]
```

Semantics

The options may appear in any order.

The FILE(<file-name>) option specifies the file involved. If it refers to a file that is not open, the file is opened implicitly.

The KEY(<scalar-expression>) option must appear if the file is a DIRECT UPDATE file, and it cannot appear otherwise. The expression is converted to a character string and determines which record is written.

The FROM(<variable>) option may be given to specify an unsubscripted level 1 variable which is to be used as the source for the record. The FROM(<variable>) option must be specified for a DIRECT UPDATE or SEQUENTIAL UNBUFFERED UPDATE file. The FROM option can be omitted for SEQUENTIAL BUFFERED UPDATE files only, in which case, the file is updated from the buffer associated with the file.

To access such a file, the sequence of statements should normally be READ followed by REWRITE.

For SEQUENTIAL UPDATE files, consecutive rewrites without reads increment the record pointer.

Example

```
REWRITE FILE (ALPHA);
```

The last record read from the data set associated with file ALPHA is rewritten from the buffer.

7.3.6.11 The WRITE Statement

The WRITE statement transfers the contents of a variable in internal storage to a record in a RECORD file.

Syntax

```
<write-statement> ::=WRITE <option-list>;
<option-list> ::=
    [FILE(<file-name>)] [FROM (<variable>)]
    [PAGE] [SKIP[(<scalar-expression>)]]
    [INDEX(<scalar-expression>)]
    [KEYFROM(<scalar-expression>)]
    [SUBFILE(<scalar-expression>)]
    [DONTWAIT(<scalar-expression>)]
```

Semantics

The file referenced must be either a RECORD OUTPUT or a DIRECT RECORD UPDATE file.

The options may appear in any order.

The FILE (<file-name>) option specifies the file in which the record is to be written. If this option is omitted, SYSPRINT is assumed. When this statement is executed the specified file is implicitly opened if it was not open at that time. If the system file attribute MYUSE is not already 'UPDATE', it is changed to 'OUTPUT'.

The FROM (<variable>) option specifies the unsubscripted level 1 variable from which the record is to be written.

The PAGE option can be specified for printer files only. The effect of this option is to advance the paper to the top of the following page before writing the record.

The SKIP option causes a new current line to be defined within the file. The expression, if present, is converted to an integer, W, which for non-print files must be greater than zero. The file is positioned at the start of the W-th line (i.e., record) relative to the current line. If the expression is omitted, SKIP(1) is assumed.

For printer files, W may be less than or equal to zero. In this case, the effect is that of a carriage return with the same current line. If less than W lines remain on the current page when a SKIP(W) is issued, the ENDPAGE condition is raised.

The INDEX option causes a record to be written at a specific location in a tape or disk file. The expression is converted to an integer, W, which must be greater than or equal to zero. The file is positioned at the start of the W-th record in the file before the write is performed, i.e., file is zero relative.

The expression in the KEYFROM option is converted to a character string and associated with the record as its key.

A record written to a line printer file may employ special carriage control characters in the first character position of the record. This character will control paper positioning before or after the record is written without use of the PAGE or SKIP option. The details for this facility will be specified after system file attributes are specified.

The SUBFILE and DONTWAIT options may be specified only for PORT files.

The value of <scalar-expression> in a SUBFILE clause specifies the subfile in a PORT file to which the WRITE statement writes. If this value is zero, a WRITE statement may write to all subfiles.

The <scalar-expression> in the DONTWAIT clause is converted to type BIT(1). If the resulting value is '0'B (or if the DONTWAIT clause is absent), program execution waits until the write is finished. If the resulting value is '1'B, a write is done only if a buffer is available. If <scalar-expression> is a variable or pseudovariable:

1. '1'B is stored in the variable or pseudovariable if the write was not done because no buffer was available.

2. '0'B is stored in the variable or pseudovariable if the write was done.

If a WRITE statement with a SUBFILE clause or a DONTWAIT clause is executed for a non-PORT file, an UNDEFINEDFILE condition results.

Examples

1. WRITE FILE(BETA) FROM(UPDATE) KEYFROM(ONKEY);

   Specifies that the record UPDATE be written as the next record in the file BETA. The key identifying the record in the file is taken from ONKEY.

2. WRITE PAGE FILE(SYSPRINT) FROM(BUF);

   Specifies that the record BUF be written as the next record in the printer file SYSPRINT after advancing to the top of the page.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

7.3.8 System Attribute Statements

7.3.8.1 The System File Attribute Assignment Statement

The <system-file-attribute-assignment-statement> is used to set the value of a system file attribute for a file outside of a file declaration or an OPEN statement.

Syntax

```
<system-file-attribute-assignment-statement> ::=
     <system-file-attribute> (<file-designator>) =
     <scalar-expression>;
```

Semantics

Attempting to set a read-only attribute will generate a syntax error of level three (3) at compile-time and a diagnostic error at run-time.

Multiple assignments are not allowed.

If a file attribute has mnemonics they must be used. This can be done by specifying a string constant or a string variable which has been assigned the correct value. If a constant is used a compile-time check is made to see if it is a valid mnemonic for the specified file attribute. If not an error of level three is given. If a variable is used it is checked at run-time and if it is an invalid mnemonic, a non-fatal run-time error is given.

7.3.8.2 The System File Attribute Reference Statement

The <system-file-attribute-reference-statement> is used to reference the value of a system file attribute.

Syntax

```
<system-file-attribute-reference-statement> ::=
     {<scalar-variable> | <pseudo-variable>} =
          <attribute-name> (<internal-file-name>
          [,<scalar-expression> [,<scalar-expression>]]);
```

Semantics

The two <scalar-expression>s are interpreted as follows: if KIND = PORT or REMOTE, the first <scalar-expression> identifies the STATION number, and the second <scalar-expression> is not allowed. If KIND = DISK or PACK, one <scalar-expression> refers to row number; two <scalar-expression>s refer to row number followed by copy number.

An attempt to read a write-only attribute generates a syntax error of level three at compile-time and a diagnostic error at run-time.

7.3.8.3 The System Task Attribute Assignment Statement

The <system-task-attribute-assignment-statement> is used to set the value of a system attribute for a task.

Syntax

```
<system-task-attribute-assignment-statement> ::=
     <system-task-attribute> (<task-designator>) =
     <scalar-expression>;
<task-designator>::= <task-variable>
```

Semantics

Attempting to set a read-only attribute will generate a syntax error of level three (3) at compile-time and a diagnostic at run-time.

Multiple assignments are not allowed.

If a task attribute has mnemonics they may be used if desired, but mnemonics are not required.

7.3.8.4 The System Task Attribute Reference Statement

The <system-task-attribute-reference-statement> is used to reference the value of a system task attribute.

Syntax

```
<system-task-attribute-reference-statement> ::=
     {<scalar-variable> | <pseudo-variable>} =
          <system-task-attribute> (<task-designator>);
```

Semantics

Attempting to read a write-only attribute will generate a syntax error of level three (3) at compile-time and a diagnostic error at run-time.

7.3.9 The Null-Statement

The <null-statement> is an empty statement. The <null-statement> causes no action and does not affect sequential operations in any way.

Syntax

```
<null-statement> ::= ;
```

Example

```
          .
          .
          .
ON OVERFLOW;
          .
          .
          .
```

The <on-unit> is a <null-statement>.

SECTION 8. INPUT/OUTPUT

A collection of data external to the program constitutes a file. Input activity transmits data from a file to a program. Output activity transmits data from a program to a file. Data transmission statements refer to a file name declared in the program and associated with a file.

In STREAM data transmission, the file can be considered to be a continuous stream of characters. The GET and PUT statements are used to transmit data values from and to the file. Conversions may occur during transmission. (See Section 8.4.2, Modes of Stream Transmission.)

In RECORD data transmission, the file consists of discrete records. The READ and WRITE statements cause a single record to be transmitted from or to the file. Transmission is direct, without any conversion, either directly to or from data variables or an intermediate buffer that may be addressable. When transmission is to or from data variables, the attributes of the variables should accurately describe the composition of the record accessed.

8.1 FILE ATTRIBUTES

The present section describes how attributes are collected and become associated with a file.

The file attributes can be divided into two categories: alternative attributes and additive attributes. Alternative attributes are those in which one group may be selected. If there is no explicit or implied declaration for one of the alternatives, and if one of those alternatives is required, a default attribute is selected. Additive attributes are those that are never applied by default and must always be stated explicitly either in a file declaration or in the OPEN statement. The exceptions are that KEYED is implied by DIRECT, and PRINT may be supplied for the SYSPRINT file. (See Section 8.4.1, Stream Transmission Statements.)

The use of PORT file attributes in PL/I is similar to that in other languages. PORT file attributes are documented in the B 7000/B 6000 Series I/O Subsystem Reference Manual.

Alternative attributes and their defaults:

ATTRIBUTES	DEFAULT
STREAM | RECORD	STREAM
INPUT | OUTPUT | UPDATE	INPUT
SEQUENTIAL | DIRECT	SEQUENTIAL
INTERNAL | EXTERNAL	EXTERNAL

Following is a list of the additive attributes:

PRINT
KEYED
PORT
{ENVIRONMENT | OPTIONS}(<file-attribute-list>)

8.1.1 Merging of Attributes

There may be a conflict between the attributes specified in a file declaration and the attributes merged as the result of the file opening, either explicit or implicit. For example, the attributes INPUT and UPDATE are in conflict, as are the attributes UPDATE and STREAM.

When a conflict occurs between merged attributes, the last specified attribute is given precedence.

After the attributes are merged, the attribute implications, listed below, are applied prior to the application of default attributes discussed earlier in this section. Implied attributes can also cause a conflict. If a conflict of attributes exists after the application of default attributes, the latest attributes have precedence.

Attributes and their implied attributes:

MERGED ATTRIBUTE	IMPLIED ATTRIBUTE(S)
UPDATE	RECORD
SEQUENTIAL	RECORD
DIRECT	RECORD,KEYED
PRINT	OUTPUT,STREAM
KEYED	RECORD

The following two examples illustrate attribute merging for an explicit opening and for an implicit opening:

Example Explicit Opening

```
DECLARE LISTING FILE STREAM;
    .
    .
    .
OPEN FILE (LISTING) PRINT;
```

The DECLARE statement establishes the file name LISTING as EXTERNAL by default, before execution commences.

Attributes after merge, due to execution of the OPEN statement, are EXTERNAL, STREAM and PRINT.

Attributes after implication are EXTERNAL, STREAM, PRINT, and OUTPUT.

This is a complete set of file attributes. No file attribute defaults are applied. The default attribute SEQUENTIAL does not apply as this attribute can be specified only for RECORD files.

The <exponent> is an integer constant. If the exponent and the preceding E or sign are omitted, a zero exponent is assumed.
Output Semantics

The data item in the data field has the following general form:

If D < S:

[-] <S-D digits> . <D digits> E{+|-} <exponent>

If D = S:

[-] 0. <D digits>{E +|-} <exponent>

At least one nonfractional digit always appears. The <exponent> is an integer of five digits. When D < S, the exponent is adjusted so that the leading digit of the mantissa is nonzero. When D = S, the exponent is adjusted so that one zero digit appears before the point of the mantissa. In the case of the value zero, one zero digit appears before the point and D zero digits after the point. The exponent is zero and its associated sign is +.

If the above form does not fill the field of width W, it is right-adjusted and the blanks are inserted on the left. If the field width W is such that low-order digits are removed, the remaining lowest-order digit is rounded as if it was followed by a digit greater than or equal to 5.

If S is omitted, it is taken as equal to D + 1. When D < S, the field width W must be greater than or equal to S + 8 for non-negative values, and to S + 9 for negative values, of the data item. However, if D is zero, the decimal point is not transmitted, and W is equal to S + 7. If the length of the data item is greater than W, the SIZE condition results.

8.4.6.2.3 Numeric Picture Format Item

Numeric data may be described by a numeric picture format item.

Syntax

<numeric-picture-format-item> ::=

P'<numeric-picture-specification>'

Semantics

The <numeric-picture-specification> is described in Section 4.3.2, Numeric Picture Data Description.

On input, the form of the data in the data stream must exactly match the picture specification.

On output, the value of the list item is converted to coded arithmetic type and the converted value is edited to the form specified by the picture specification before it is transmitted.

8.4.6.2.4 Bit String Format Items

The bit string format item describes the representation of a bit string in the data stream.

Syntax

```
<bit-string-format-item> ::=

     B [(W)]
```

Semantics

In the case of input, W is always required. For output, if W is omitted, it is taken to be the current length of the associated bit string or the length obtained by converting the item to a bit string.

On input, the data field transmitted is a character representation of a bit string anywhere within the field of width W.

On output, the value of the list item is converted to bit and the resulting bit string is left-adjusted in the field of width W. Truncation, if necessary, is performed on the right. Zeros are used for padding.

8.4.6.2.5 Character String Format Items

Character data may be described by a character string format item.

Syntax

```
<character-string-format-item> ::=

     { A [(W)]} |
     { P'<character-picture-specification>'}
```

STRING String Built-In Function

Definition: STRING concatenates all the elements in the result of an expression into a single string element.

Reference: STRING (X)

Arguments: The argument, X, is an element, array, or structure expression, the result of which is composed either entirely of character strings and/or decimal numeric character data, or entirely of bit strings and/or binary numeric character data.

Result: The value returned by this function is an element bit string or character string, the concatenation of all the elements in X. If X contains one or more varying strings, the result is a varying string.

SUBSTR String Built-In Function

Definition: SUBSTR extracts a substring of user-defined length from a given string and returns the substring to the point of invocation. (SUBSTR can also be used as a pseudo-variable.)

Reference: SUBSTR (<string>, I [,J])

Arguments: The argument <string> represents the string from which a substring will be extracted. If this argument is not a string, it will be converted, before the function is applied, to a character string if the argument is decimal, or to a bit string if the argument is binary. Argument I represents the starting point of the substring and J represents the length of the substring. Arguments I and J must be integers or expressions that can be converted to integers.

Assuming that the length of <string> is K, arguments I and J must satisfy the following conditions:

1. J must be less than or equal to K and greater than or equal to 0.

2. I must be less than or equal to K and greater than or equal to 1.

3. The value of I+J-1 must be less than or equal to K.

Thus, the substring, as specified by I and J must lie within <string>.

If J is not specified, it is assumed to be equal to the value of K-I+1. In other words, it is assumed to be the length of the remainder of <string>, beginning at the I-th position in <string>.

When these conditions are not satisfied, the SUBSTR reference causes the STRINGRANGE-interrupt to be raised, if it is enabled.

Result: The value returned by this function is a varying-length string whose current length is defined as follows:

1. If J=0, the returned value is the null string.

2. If J is greater than 0, the returned value is that substring beginning at the I-th character or bit of the first argument and extending J characters or bits.

3. If J is not specified, the returned value is that substring beginning at the I-th character or bit and extending to the end of <string>.

TRANSLATE String Built-In Function

Definition: TRANSLATE forms a character string whose value is the result of applying a given translation to a given character string, and returns the result to the point of invocation.

Reference: TRANSLATE (S,R[,M])

Arguments: The first argument, S, represents the source character string to be translated. The arguments R and M control the translation. Arguments that are not strings are converted to character strings. If R is shorter than M, it is padded on the right to the length of M with blanks. If M is omitted, COLLATE( ) is assumed as the value of M.

Result: The value returned by this function is a character string whose length is that of S. The value r of each character in the result is related to the character c in the corresponding position in S by the following formula:

r=SUBSTR(R, INDEX(M,c),1)

unless INDEX(M,c) is 0, in which case r=c.

Thus, the string S is searched character by character until a character is found which is a member of the string M. When such a character is found it is replaced by the character in R appearing in the position in R corresponding to its position in M.

For example, the result of TRANSLATE('5206900','ADGJMPTW',23456789') is 'JA0MW00'.

UNSPEC String Built-In Function

Definition: UNSPEC returns a bit string that is the internal coded representation of a given value. (UNSPEC can also be used as a pseudo-variable.)

Reference: UNSPEC (X)

Argument: The argument, X, may be an arithmetic, string, locator, or area expression, or an area variable, whose internal coded representation is to be found.

Result: The value returned by this function is the internal coded representation of X. This representation is in bit-string form, the length of this string depends upon the attributes of X.

APPENDIX 2. CONDITIONS

The conditions are those conditions that may be specified in the ON statement. These conditions are also specified in SIGNAL and REVERT statements.

For each condition name, the description in this appendix includes the circumstances under which the condition occurs, the standard system action that would be taken in the absence of programmer-specified action, and, where applicable, the result.

For conditions OVERFLOW, UNDERFLOW, ZERODIVIDE, CONVERSION, or FIXEDOVERFLOW, an interrupt action will always take place on occurrence of the condition, unless the occurrence is in a calculation lying within the scope of a prefix specifying NOOVERFLOW, NOUNDERFLOW, NOZERODIVIDE, NOCONVERSION, or NOFIXEDOVERFLOW. For the conditions SIZE, STRINGRANGE, SUBSCRIPTNAME, or CHECK <identifier list>, an interrupt will not take place on occurrence of the condition unless the occurrence is in a calculation lying within the scope of a prefix specifying the condition. (See Section 10.5.2, Scope of the Condition Prefix.)

For any other condition whose name may not be used in a prefix, an interrupt always will result from the occurrence of the condition.

A multiple interrupt can occur only for an input/output operation that has been associated with an event variable, and is described in the section of this appendix dealing with input/output conditions.

CLASSIFICATION OF CONDITIONS

The conditions are classified as follows: computational, input/output, program-checkout, list processing, programmer-named, and system-action conditions.

The computational conditions are associated with data handling, expression evaluation, and computation.

The input/output conditions are associated with data transmission.

The program-checkout conditions facilitate debugging of programs.

The list processing conditions are associated with area usage.

The programmer named conditions permit the programmer to use conditions of his own naming. These conditions are raised only by a SIGNAL statement.

The system-action conditions provide facilities to the programmer to extend the standard system action taken after the occurrence of a condition or at the completion of a program.

CONVERSION:

This condition is raised whenever an illegal conversion is attempted on character string data, either internally or during input or output. The condition will be raised for such errors as characters other than 0 or 1 in conversion to bit string, characters not permitted in conversion to numeric field, or illegal characters in conversion to arithmetic. The conversion is carried out character-by-character. The condition is raised for each illegal conversion.

This condition may also be raised when the number of digits in a floating-point exponent exceeds the number allowed. On return from the on-unit for this condition, the conversion will be reattempted.

Expressions that are not variables, or variables that are procedure entry parameters, based, or overlay-defined, may not be reevaluated after an on-unit. Therefore, changing the value of such an expression in an on-unit may not affect the conversion retry.

Result: When CONVERSION occurs, results of the entire result field are undefined.

Standard System Action: Comment and raise the ERROR condition.

FIXEDOVERFLOW:

This condition occurs during fixed-point arithmetic operations if the results of these operations exceed the integer capacity of the hardware. (See SIZE below for a related condition that checks for overflow related to declared attributes of data items.)

Result: Result of the invalid fixed-point operation is undefined.

Standard System Action: Comment and raise the ERROR condition.

OVERFLOW:

This condition occurs when the exponent of a floating-point number exceeds the permitted maximum.

Result: The value of such an invalid floating-point number is undefined.

Standard System Action: Comment and raise the ERROR condition.

SIZE:

This condition is raised by conversions between data types, or between differing BASES, SCALES, or PRECISIONS. The condition arises if the high order bits or digits are lost when a value is assigned to an arithmetic data item, or when a value is converted during expression evaluation or during input/output.

The SIZE condition should be distinguished from FIXEDOVERFLOW that occurs during arithmetic calculations. A value too large for the field to which it is assigned will raise a SIZE condition on assignment, regardless of whether there was a FIXEDOVERFLOW in the calculation of the value.

Result: The contents of the receiving field are undefined.

Standard System Action: Comment and raise the ERROR condition.

	1608	Bit string truncated in string to arithmetic conversion.
	1609	"E" format field width too small, E(N+9,N).
AREA		
	1701	Allocate error (allocate of negative size requested).
	1702	Free error (chunk is already free).
	1703	Invalid allocation intrinsic parameter.
	1704	Allocate error (area is exhausted).
	1705	Attempted to free a non-freeable area.
	1706	Allocation size error on unit-size area.
	1707	Free error on unit-size area.
	1708	Free error (invalid free stacked).
	1709	Allocate error (invalid area header).
	1710	Free error (invalid area header).
	1711	Free error (invalid link word).
STRINGSIZE		
	1801	Source string is longer than target string in bit string to character string conversion.
	1802	Source string is longer than target string in character string to bit string conversion.
	1803	Source string is longer than target string in bit string to bit string conversion.
	1804	Source string is longer than target string in character string to character string conversion.
CONVERSION		
	2101	Conversion error on "E" format item input.
	2102	Illegal binary character in string to string conversion.
	2103	Illegal character in string to arithmetic conversion.
	2104	Illegal binary character in string to arithmetic conversion.
	2105	Illegal imaginary string in string to arithmetic conversion.
	2106	Illegal imaginary bit string in arithmetic conversion.
	2107	Invalid character for numeric picture editing.
	2108	Invalid character for character picture editing.
	2109	Invalid character for corresponding picture character.
	2110	Error in string to arithmetic conversion.
	2111	Missing mantissa in string to arithmetic conversion.
	2112	Missing exponent following "E" in floating point number.
	2113	Missing "I" following complex number.
KEY		
	2401	No record was found with this key.
	2402	No space is available for additional keyed records.
	2403	Attempted to write a duplicate keyed record with the NODUPLICATE option set.
	2404	A duplicate keyed record was added.
	2405	A difference in key exists on a REWRITE statement.
	2406	A key error exists on a CREATE write. For example, the keys are out of order.
	2407	Table overflow on a KEYED CREATE WRITE. Check areasize.
	2408	A REWRITE was attempted before a READ was executed.
	2409	Invalid usage of KEYED FILE or KEYED OPTION.
	2410	Write KEYFROM key and record key do not match.

2411	Record contains '11111111' in first byte.
2412	KEYTO variable shorter than key in file.
2413	Key or KEYFROM longer than key in file.

UNDEFINEDFILE

2501	Attribute conflict on this file.
2502	Illegal file access method.
2503	Attempt to open a non-resident file.
2504	Attempted to close an unopened file.
2505	Parameter error on KEYED FILE OPEN.
2506	KEYED FILE opened incorrectly.
2507	Error occurred while opening a KEYED file.
2508	Cannot handle a bcl file.
2509	Attempted to open a previosly opened file.
2510	Attempted a keyed I/O on a non-keyed file.
2511	Keyed file not declared direct.
2512	This I/O action illegal for keyed files.
2513	Open attempted on a non-present file.
2514	Rewrite attempted on variable length records.
2517	Cannot create empty keyed file.
2518	Bad subfile index.
2519	Read or write to subfile failed.

ERROR

2601	An attempt was made to access an unititialized variable.
2602	Unimplemented or unsupported format phrase.
2603	Format error, missing left parenthesis.
2604	SQRT error.
2605	Picture conversion error was not corrected.
2606	Double SQRT error.
2607	String to arithmetic conversion error was not corrected.
2608	String to string conversion error was not corrected.
2609	Arithmetic overflow occurred during conversion.
2610	Arithmetic overflow occurred in arithmetic to character conversion.
2611	No data format item specified.
2612	Explicit field width not specified in format.
2613	Invalid LNGAMMA argument.
2614	Invalid LOG argument.
2615	Invalid ARCCOS argument.
2616	Invalid ARCSIN argument.
2617	Invalid SINH OR COSH argument.
2618	Invalid GAMMA argument.
2619	Invalid SQUARE ROOT argument.
2620	Invalid list item in GET STRING data.
2621	End of input in GET STRING statement.
2622	End of output in PUT STRING statement.
2623	Illegal format phrase with string edit.
2624	Invalid format for "R" format phrase.

DMEXCEPTION

2701	Data management error.

Controls the sorted listing keyed by the number of times that each block and procedure was called in the program execution. The meaning of '=integer' is the same as described for the TOTAL option, above.

Any of the TOTAL, AVERAGE or COUNT options that do not appear will default to TOTAL=0, AVERAGE=0, or COUNT=0 as appropriate. If the STATCONTROL option is not set in a compilation that also sets the STATISTICS option, a default of

```
STATCONTROL ( TOTAL=0  AVERAGE=0  COUNT=0 )
```

will be used; this will cause only the unconditional statistics printer listing to be produced.

The compiler "LIST2" listing will indicate the beginning of each program "logic unit" when the LOGIC or COUNTS specifications of the STATISTICS option is SET. The form of this indication is:

```
procedure-name : lll
          or
   BLOCK nnnn : lll
```

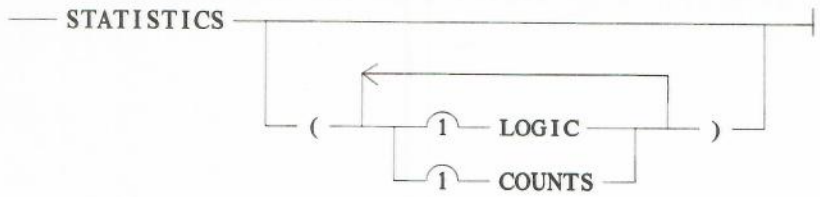
The "lll" represent the number of the logic unit for the indicated procedure or block, and are used to identify the entries in the statistics summary listings produced following program execution.

STATISTICS

Type: boolean

Default: disabled

Syntax



This control performs the normal SET/RESET/POP operations on the compiler toggle that indicates whether or not to compile statistics gathering information for procedures and BEGIN-END blocks. This option returns the processor times from the execution of these sections of code. If the control is enabled when a PROCEDURE or a BEGIN statement is encountered, code will be compiled for that entire procedure or block to gather timing statistics. If the STATISTICS option is disabled at the beginning of a procedure or block, no statistics code will be generated for that procedure or block. If the option is enabled later within the procedure or block, only subsequent procedures and blocks will gather statistics.

LOGIC

Indicates that additional timing statistics are to be gathered for the program's individual logic units. A logic unit is the stream of executable code that begins at a point that can be branched to and ends at the next such "branch point". This is not limited to explicit program labels, as the compiler may generate "branch points" for various language constructs (such as IF...THEN...; ELSE...; and repetitive DO statements). Injudicious use of the LOGIC option can cause a program to run noticably slower. RESET STATISTICS also disables LOGIC and POP STATISTICS pops to the previous setting for both STATISTICS and LOGIC.

COUNTS

Is like the LOGIC specifier in that it gathers information about individual logic units within a program. The only difference between COUNTS and LOGIC is that COUNTS collects only usage counts for the logic units (elapsed time information is only collected for procedures and blocks). Injudicious use of the COUNTS option can cause a program to run noticably slower (although not as seriously as with the LOGIC specifier). RESET STATISTICS also disables COUNTS and POP STATISTICS pops to the previous setting for both STATISTICS and COUNTS.

STMTNO

Type: boolean

Default: enabled

Functional Description

This option, when enabled, causes information which associates a symbolic image sequence number with program addresses to be stored with the object program.

Syntax

Semantics

1. This option causes sufficient information to be stored with the object program to permit the source language sequence number that is associated with a given program address to be determined.

2. Should a program terminate abnormally, the source language sequence number associated with the point of termination is provided.

Syntax

Semantics

1. The summary produced by the enabling of this option is the same as that produced if the "LIST1 or LIST2" option is true. Therefore, this option takes effect only if the "LIST" options are all disabled.

TITLE

Type: value

Default: the title of the code file being generated.

Functional Description

This option causes the compiler to store a title to be used on all subsequent pages of the output listing.

Syntax

Syntax Rules

1. The 'alphanumeric literal' must contain at least one character.
2. The maximum number of characters in 'alphanumeric literal' is 66.
3. The quote character (') may not appear within the 'alphanumeric literal'.

Semantics

1. The compiler will store the specifed 'alphanumeric literal' for use as a title on all subsequent pages of the output listing.
2. Subsequent uses of this option result in the storing of the new 'alphanumeric literal' and the previous title is lost.

TRACE

Type: boolean

Default: disabled

Functional Description

This option, when enabled or disabled, enables or disables the "TRACEENTRYS" and the "TRACELABELS" options.

Syntax

TRACEENTRYS

Type: boolean

Default: disabled

Functional Description

This option, when enabled, causes the object program to note on the SYSPRINT file each entry into a procedure.

Syntax

TRACELABELS

Type: boolean

Default: disabled

Functional Description

This option, when enabled, causes the object program to note on the SYSPRINT file, each labelled statement that is executed.

Syntax