Burroughs

PUBLICATION CHANGE NOTICE

PCN No.: 5001530-003 Date: March 1, 1982

Publication Title: B 5000/B 6000/B 7000 Series PL/I Language Reference Manual (May 1977)

Other Affected Publications:

Supersedes: N/A

Description:

Incorporates the Mark 3.3 System Software release.

Replace these pages	Add pages
TOC-1	4-36-A
TOC-3	7-14-A
TOC-5	7-24-A
TOC-7	
TOC-9	
TOC-11	
4-35	
7-13	
7-23	
7-53	
7-55	
9-11	
Index-1	
Index-3	
Index-5	
Index-7	
Index-9	
Index-11	
Index-13	

TABLE OF CONTENTS

Assumption:

The default <scope-attribute> is INTERNAL for any item with the BUILTIN attribute.

4.7.4 Returns Attribute

The <returns-attribute> may be specified in a DECLARE statement for an entry name that is used as a function reference within the scope of the declaration.

Syntax

<returns-attribute> ::= RETURNS (<attribute-list>);

Semantics

The <returns-attribute> specifies the attributes of the function value returned when the entry name is invoked as a function.

The attributes in the <attribute-list> must agree with the attributes specified explicitly or by default in the PROCEDURE statement or ENTRY statement to which the entry name is prefixed.

The <length-attribute> specifications are evaluated on entry to the block containing the RETURNS attribute specification. Such evaluated <returns-attribute>s form part of the environment of blocks contained within the block declaring the attribute and dynamically descendant from the block.

Restrictions:

For an internal function, the <returns-attribute> can be specified only in a DECLARE statement that is internal to the same block as the function procedure.

Only arithmetic, string, locator, picture, aligned and unaligned attributes can be specified in the attribute list.

Assumptions:

If an entry name has no <returns-attribute> specification, a RETURNS attribute is assumed.

If the entry name begins with any of the letters I thru N, the defaults are REAL, FIXED, BINARY with default precision. If the entry name begins with any other letter, the defaults are REAL, FLOAT, DECIMAL with default precision.

4.7.5 Parameter Attribute

The <parameter-attribute> denotes a data item that is a parameter to an entry point of a procedure.

Syntax

<parameter-attribute> ::= [PARAMETER]

Semantics

This attribute is explicitly specified for a variable by the appearance of the name of that variable in a <parameter-list> and does not need to be specified by the programmer.

4.7.6 Library Attribute

A library attribute specifies that the identifier being declared is a <library-identifier>.

Syntax

```
<library-attribute> ::=

     LIBRARY [(<system-library-attribute-list>)]

<system-library-attribute-list> ::=

     <system-library-attribute>
     [,<system-library-attribute>]...

<system-library-attribute> ::=

     INTNAME = <char-string-constant>
     TITLE = <char-string-constant>
     LIBPARAMETER = <char-string-constant>
```

Semantics

The TITLE, INTNAME and LIBPARAMETER attributes are EBCDIC string-type attributes. The INTNAME is the internal identifier for the library. The TITLE is the code file name of the library. The LIBPARAMETER transmits information from the user program to the selection procedures of library programs that provide dynamic entry point linkage.

The character strings specified for system library attributes must be constants. Specification of these attributes is optional; by default, the identifier being declared is used for the TITLE and the INTNAME. If the INTNAME is specified but the TITLE is not, the string specified for the INTNAME is used for the TITLE.

4.7.7 Options Attribute

The OPTIONS attribute defines an entry as an external library entry point.

Syntax

```
<options-attribute> ::=

     ENVIRONMENT/OPTIONS (LIBRARY=<library-identifier>

    [,ACTUALNAME=<char-string-constant>])
```

Semantics

Use of an OPTIONS list that includes LIBRARY or ACTUALNAME implies the ENTRY and CONSTANT attributes for the identifier being declared. LIBRARY and ACTUALNAME are mutually exclusive with file attributes; this use implies that a file is being declared. The library must be named; the declared <library identifier> is used by default. The specification of an ACTUALNAME is optional.

Example

In this example, calling the entry DO-THIS from the PL/I program causes the MCP to link the program to a library called YOUR-LIBRARY with an entry point called DO-THAT.

```
DECLARE LIBI LIBRARY
     (TITLE = "MATH/LIBRARY");

DECLARE MY LIBRARY LIBRARY
     (INTNAME = "YOUR-LIBRARY");

DECLARE DO-THIS ENTRY (BINARY FIXED, CHAR(*))
     RETURNS (CHARACTER(100) VARYING)
     OPTIONS (LIBRARY = MY-LIBRARY,
              ACTUALNAME = "DO-THAT");
```

4.8 FILE DATA DESCRIPTION

Syntax

```
<file> ::=

     <attribute> |

     <function-attribute> |

     <transmission-attribute> |

     <print-attribute> |

     <key-attribute> |

     <access-attribute> |

     <environment-attribute>
```

4.8.1 File Attribute

The <file-attribute> specifies that the identifier being declared is a file name.

Syntax

<file-attribute> ::= FILE

Assumptions:

The <file-attribute> can be implied by any of the other file description attributes.

An identifier may be contextually declared with the <file-attribute> through its appearance in the <file-option> of any data transmission statement or in an ON statement for any data transmission condition.

The <scope-attribute> EXTERNAL is assumed for files that are not parameters.

represented by <expression5> through <expression8>, the second expansion would look like this:

```
NEXT:   E5 = <expression5>;
                .
                .
                .
           V=E5;
LABEL3:    IF ... THEN GO TO NEXT1;
           IF (<expression8> THEN;
             ELSE GO TO NEXT1;
           statement 1
                .
                .
                .
           statement m
LABEL4:    V=V+E7;
           GO TO LABEL3;
NEXT1:  statement n
```

2. If the <while-expression> is omitted, the IF statement immediately preceding statement 1 in the expansion is omitted.

3. If TO <expression2> is omitted, the statement E2 = <expression2> and the IF statement identified by LABEL2 are omitted.

4. If both TO <expression2> and BY <expression3> are omitted, all statements involving E2 and E3, and the statement GO TO LABEL2 are omitted.

5. Although the above expansions show a specific order in which the BY and TO-clauses are evaluated, no specific ordering is defined by the language.

The <while-expression> specifies that before each associated execution of the do-group, the expression is evaluated and, if necessary, converted to give a bit-string value. If any bit in the resulting string has the value '1', the iteration continues. If all bits have the value '0', the iterations associated with the current specification are terminated.

In the <iteration-specification> list, <expression1> represents the starting value of the control variable, <expression3> represents the increment to be added to the control variable after each iteration of the statements in the do-group. <expression2> represents the terminating value of the control variable. Iteration terminates as soon as the value of the control variable passes its terminating value. When the last specification is completed, control passes to the statement following the do-group.

Control may, under any circumstances, be transferred into a do-group from outside the do-group if iteration is not specified, i.e., no <while-expression> or <iteration-expression> exists. If the do-group is iterative and a GO TO statement transfers control to a statement inside the group, the results are undefined unless the GO TO is an abnormal return from a block that has been activated from within the do-group.

The effect of allocating or freeing the control variable is undefined.

Examples

```
1   DO INDEX = Z WHILE (A>B), 5 TO 10 WHILE (A=B), 100;
2.  DO I = 1 TO 9,11 TO 20;
3.  DO WHILE (P);
4.  DO;
5.  DO WHILE (TAX-DEDCT < ESTTAX * 4);
6.  DO I = 1 TO 100 BY 3;
7.  DO INDEX = V REPEAT(Z) WHILE(X<100);
```

7.3.2.4 The EXIT Statement

Syntax

```
<exit-statement> ::= EXIT;
```

Semantics

The EXIT statement causes immediate termination of the task that contains the statement and all tasks attached by this task. The execution of the EXIT statement in a major task is equivalent to a STOP statement.

7.3.2.4A The FREEZE Statement

Syntax

```
<freeze-statement> ::=FREEZE OPTIONS(<freeze-option>);
<freeze-option>::= PERMANENT/TEMPORARY
```

Semantics

A program must be "frozen" at run time using the FREEZE statement in order to become a library. The FREEZE statement must occur in the outer block of the program.

7.3.2.5 The GO TO Statement

The GO TO statement causes control of a program to be transferred to the specified statement within the program.

Syntax

```
<go-to-statement> ::=
    {{GO TO}|GOTO} <statement-label-name>;
```

Semantics

If a label variable is specified, the GO TO statement has the effect of a multi-way switch. The value of the label variable is the label of the statement to which control is transferred when the GO TO statement is executed.

Since the label variable may have different values at each execution of the GO TO statement, control may not always pass to the same statement. (Example 2 illustrates a GO TO statement used as a multi-way switch.)

A GO TO statement cannot pass control to an inactive block or to another task.

If control is transferred into a do-group that specifies iteration, the results are undefined unless the GO TO specifies an abnormal return from a block that has been activated from within the do-group.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES

entry names. If no <data-specification-attributes> are specified, arithmetic defaults are applied separately to each name.

Consider the Statement:

```
A:I:  ENTRY;
```

This statement is equivalent to:

```
A:    ENTRY;
I:    ENTRY;
```

An ENTRY statement cannot be internal to a begin block or to a do-group that specifies iteration.

A condition prefix cannot be prefixed to an ENTRY statement.

7.3.3.4 The PROCEDURE Statement

Syntax

```
<procedure-statement> ::=
    <entry-name>: ...PROCEDURE
        [(<parameter> [,<parameter>]...)]
        [OPTIONS (<option-list>)]
        [RECURSIVE]
        [RETURNS (<data-specification-attributes>)];

<option-list> ::= <option> [,<option>]...

<option>::=MAIN / DOUBLE / SORTINPUT / SORTOUTPUT /

    EXPORT (<entry-name>[AS<char-string-constant>]

    [,<entry-name>[AS<char-string-constant>]]...)
```

Semantics

The OPTIONS, RECURSIVE, and RETURNS options may appear in any order.

The PROCEDURE statement has the following functions:

1. Identifies a portion of program text as a procedure.
2. Defines the primary entry point to a procedure.
3. Specifies the parameters for the primary entry point.
4. Defines any special attributes of the procedure.
5. Specifies the attributes of the value that is returned if the procedure is invoked as a function at the primary entry point.

The <parameter>s are names that specify the parameters of the entry point. When the procedure is invoked, a relationship is established between the arguments of the invocation and the parameters of the invoked entry point. (See Section 9.5, Correspondence of Parameter Arguments).

OPTIONS specifies a list of options. OPTIONS may be specified for any procedure. DOUBLE allows the use of double precision for

arithmetic calculations, and SORTINPUT and SORTOUTPUT are discussed in Section 7.3.2.8, The Sort Statement.

The EXPORT option may only be used in the main (that is, outer block) <procedure-statement>. This option specifies which entry points can be used by other programs that call the program as a library. The <entry-names> used in the EXPORT option must be declared in the outer block of the program.

RECURSIVE specifies that the procedure may be invoked recursively (i.e., it may invoke itself). RECURSIVE does not have to be specified as all procedures are recursive by default.

The <data-specification-attributes> permitted for the RETURNS attribute of a PROCEDURE statement are the arithmetic, string, AREA, OFFSET, and POINTER attributes. These attributes specify the characteristics of the value returned by the procedure when invoked as a function at the primary entry point. (This rule applies to each entry name prefixed to the PROCEDURE statement.) The value specified in the RETURN statement of the invoked procedure is converted to the specified <data-specification-attributes>.

If a PROCEDURE statement has more than one entry name, the first name is interpreted as the only label on the statement. Each subsequent entry name is interpreted as a separate ENTRY statement having an identical parameter list and the same <data-specification-attributes> as written in the PROCEDURE. This equivalence is true only after multiple closure has been resolved. Defaults for the <data-specification-attributes> are applied separately for each such ENTRY statement and for the resulting PROCEDURE statement.

Examples

1. A:I: PROCEDURE;

Is effectively the same as:

```
A: PROCEDURE;
I: ENTRY;
```

The equivalence applies only after multiple closure has been resolved.

2.
```
B: PROCEDURE;
   DCL A ENTRY RETURNS(FIXED);
         .
         .
         .
   D=A(X,Y);
   END B;
A: PROCEDURE (B,C) RETURNS(FIXED);
   .
   .
   .
   RETURN (B*C + SIN (P));
   END A;
```

If procedure A is invoked as a function as it is in procedure B, then when control is returned to B, the expression (B*C + SIN (P)); is evaluated, converted to fixed point, and the value assigned to D in procedure B.

3. PROGRAM: PROCEDURE OPTIONS(MAIN, EXPORT(E1, E2 AS "E"));

In the following example, E1 and E2 are library entry points.

```
E1:  PROCEDURE
        :
     END E1;

 A:  PROCEDURE
        :
E2:  ENTRY
        :
     END A;
```

7.3.4 Data Declaration Statements

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 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, 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.

7.3.8.5 The System Library Attribute Assignment Statement

The <system-library-attribute-assignment-statement> sets the value of library attributes outside of a library declaration.

Syntax

```
<system-library-attribute-assignment-statement>::=
<system-library-attribute> (<library-identifier>)=
<scalar-expression>;
```

Semantics

Attempting to set a read-only attribute generates a syntax error at compile time and a diagnostic error at run time.

Multiple assignments are not allowed.

7.3.8.6 The System Library Attribute Reference Statement

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

Syntax

```
<system-library-attribute-reference-statement>::=
 <scalar-variable>/<pseudo-variable> =
<system-library-attribute>(<library-identifier>);
```

Semantics

The values specified for the system library attributes are subject to some constraints:

1. The TITLE must be a valid file title as judged by the system; "*", usercode, and ON PACKNAME specifications are allowed.

2. The INTNAME must be a single identifier.

3. The LIBPARAMETER may be any string and is limited only by the conventions of the particular library.

Attempting to read a write-only attribute generates a syntax error 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>.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES

9.8 LIBRARIES

9.8.1 Introduction

A library is a program which provides a set of procedural "entry points" which can be called on by "user programs". Any user can create and use libraries. (A detailed discussion of the usage of libraries is found in Chapter 20 of the B 5000/B 6000/B 7000 Series System Software Operational Guide, Volume 2. Familiarity with this material is assumed in the discussion that follows.)

9.8.2 Creating Libraries

The OPTION list of the main (that is, outer block) PROCEDURE statement must include the EXPORT option. The EXPORT option specifies which entry points the library makes available to the programs that call on the library. (Refer to Section 7.3.3.4, The PROCEDURE Statement, for a more detailed discussion of the EXPORT option.)

To become a library, a program must be "frozen" at run time using the FREEZE statement. (Refer to Section 7.3.2.4A, The FREEZE Statement, for a more detailed discussion of this statement.)

9.8.3 Referencing Libraries

To call a library, the user program declares library names and specifies which procedures within the program call a given library. Libraries must be explicitly declared using the LIBRARY attribute in a DECLARE statement. (Refer to Section 4.7.6, The LIBRARY Attribute, for a more detailed discussion of this attribute.)

An entry point name is associated with a library by adding the OPTIONS attribute to an entry point declaration. (Refer to Section 4.7.7, The OPTIONS Attribute, for a more detailed discussion of this attribute.)

9.8.4 Changing Library Attributes

Library attributes may also be set and tested dynamically using the <system-library-attribute-reference-statement>. (Refer to Section 7.3.8.6, The System Library Attribute Reference Statement, for a more detailed discussion of this statement.)

9.8.5 Entry Points and their Parameters

Parameters must be passed according to certain rules. When a PL/I program calls a library procedure written in another language, the library procedure expects to receive parameters by name or by reference and not by value. When a PL/I library is called by a program written in another language, parameters must not be passed by name.

Parameters to a library entry point may be any of the following types:

```
Binary Fixed (p,q),
...q = 0, p <= 39
...q = 0, 39 < p <= 78
...q neq 0, p <= 39
...q neq 0, 39 < p <= 78

Decimal Fixed(p, q),
...q = 0, p <= 11
...q = 0, 11 < p <= 23
...q neq 0, q <= 11
...q neq 0, 11 < p <= 23

Binary Float(p),
...p <= 39
...39 < p <= 78

Decimal Float(p),
...p <= 11
...11 < p <= 23

Character(n)
Character(*)
Character(n) Varying
Character(*) Varying

Bit(n), n <= 48

Transaction record
Transaction record array
```

INDEX