ABSTRACT

This document describes the new Systems Programming Language SPL for release at PRIMOS rev 19.0.

For rev 19.0, SPL is a derivative of PL1G. SPL follows all of the language constructs of PL1-subset G with the following exceptions:

1 PL1 I/O

Because SPL is a systems programming language, PL1G and full PL1 I/O statements, I/O on conditions, and declaration attributes are NOT supported, and any attempt to use them will result in compile errors. This list includes the statements: CLOSE, DELETE, FORMAT, GET, OPEN, PUT, READ, REWRITE, and WRITE. The following I/O on conditions are illegal unless redefined by the user: ENDFILE, ENDPAGE, KEY, UNDEFINEDFILE, and UNDF. The following declaration attributes are illegal: BACKWARDS, BUFFERED, BUF, DIRECT, ENVIRONMENT, FILE, KEYED, LINESIZE, OUTPUT, PAGESIZE, PRINT, RECORD, SEQUENTIAL, SEQL, STREAM, TITLE, UNBUFFERED, and UNBUF. Also, the following options are illegal: DATA, EDIT, FILE, FROM, INTO, KEY, KEYFROM, KEYTO, LINE, LIST, PAGE, SKIP, and STRING.

2 Select Statements

SPL Supports the select statement. The format of a Select block is:

```
     Select;
       When (<if expression>) <statement>;
            .
            .
            .
      [Otherwise <statement>;]
      End;
```

Or a Select block looks like:

```
     Select (<value>);
       When (<value list>) <statement>;
            .
            .
            .
      [Otherwise <statement>;]
      End;
```

Where <statement> is defined to be any simple statement not including DECLARE, END, ENTRY, or PROCEDURE. <statement> may include a DO block or a BEGIN block of statements, or be an IF statement. <if expression> is an expression that evaluates to a BIT(1) result as in an IF statement. <value> is any expression and <value list> is either <value> or a list of <value>s separated by commas. A select block is traversed by executing each WHEN clause until a TRUE condition is found. A TRUE condition happens when the <if expression> part evaluates to '1'B or (a <value> in the When clause) = (<value> in the Select statement). If a <value> in the When clause is not of the same data type as the <value> in the Select statement, it is converted to the data type of the latter before the comparison is done. After either a When clause or the Otherwise clause is executed, control passes to the first executable statement following the Select Block. If none of the When clauses are satisfied and an Otherwise clause exists, then the otherwise clause is executed, else ERROR is signalled. NOTE: PLP does not signal an error and SPL does for a missing otherwise clause. Only one clause of a Select block will be executed per invocation.

3 UNTIL

SPL supports the Until option of a DO statement. Until is the opposite of the While option. It has the form UNTIL(<if expression>), Where <if expression> is an expression that evaluates to a BIT(1) result as in an IF statement. The result of an Until option is to execute the Do group, and then test the Until part to see if it is false. The Do group will continue to be executed WHILE the until part is false and UNTIL it is true. Until differs from the While option in that the Do group is only executed when the While is true. When the While part becomes false, execution continues after the Do group. While and Until can be used together to form a double exit from the Do group as in: DO Until(exp1) While(exp2). In this case the Do is executed when While is true. After the first execution, we check the Until part. If it is false, we repeat the While test and continue. If it is true, we leave the Do group and continue. Until can appear anywhere in a Do statement that While can appear.

4 LEAVE

Leave is a statement for leaving loops without using a GOTO statement. It is intended as a more structured approach for leaving a loop, since when executed, execution continues with the first statement after the corresponding Do group containing the Leave. Leave has The following form: LEAVE [<label>]; where <label> is optional and if present is the label of the Do group that we want to leave. For example:

```
                          acts like:
  DO WHILE('1'B);                          DO WHILE('1'B);
    IF ready_to_leave                        IF ready_to_leave
      THEN LEAVE;                              THEN GO TO EXIT;
      ELSE continue;                           ELSE continue;
   END;                                      END;
                                           EXIT:;
```

and

```
                          acts like:
  LOOP1: DO WHILE('1'B);                   DO WHILE('1'B);
     IF loop_2                                IF loop_2
       THEN DO UNTIL(done);                     THEN DO UNTIL(done);
         IF end_of_list                           IF end_of_list
           THEN LEAVE LOOP1;                        THEN GOTO EXIT1;
           ELSE continue;                           ELSE continue;
        END;                                     END /* Do Until */;
       ELSE continue;                           ELSE continue;
     END LOOP1;                             END /* Do While */;
                                           EXIT1:;
```

NOTE: If the LEAVE statement is used inside of a non-iterative usage of the DO statement, that it will still leave THAT DO, not an iterative DO that might contain the non-iterative DO. All DO's are created equal!

5 LIKE

Like is an option of a structure declaration in a DECLARE statement. It has the form: LIKE <structure reference>, where <structure reference> is the name of a structure declared earlier in the program and known to the block containing the LIKE declaration; it need not be a level-1 structure. Like has the effect as if the all of the members of <structure reference> had been copied directly following the variable with the like attribute, except that the level numbers are adjusted upward or downward as necessary to be compatible with there position in the new structure. For example:

```
Declare 1 structure_1,
          2 length Fixed;
          2 string Char(80);
Declare 1 structure_2,
          2 char_var_1 like structure_1,
          2 size Fixed,
          2 char_var_2 like structure_1;
```

is the same as:

```
Declare 1 structure_1,
          2 length Fixed;
          2 string Char(80);
Declare 1 structure_2,
          2 char_var_1,
            3 length Fixed,
            3 string Char(80),
          2 size Fixed,
          2 char_var_2,
            3 length Fixed,
            3 string Char(80);
```

6 OPTIONS(VARIABLE)

SPL contains a declaration attribute for specifying that an entry point can be called without checking its arguments for the correct type of parameters. The attribute is called OPTIONS(VARIABLE), and is used as in: DECLARE VARY ENTRY OPTIONS(VARIABLE);. This means that VARY can be called from the user's program with different arguments without causing errors, for example:

```
    CALL VARY(12345E3);
    CALL VARY('This ia a string', 23, '1'B);
    CALL VARY(VARY);
```

would then all be valid uses of the entrypoint VARY; The first will call VARY with one argument, the floating point number 12345 * 10 ** 3. The second will call VARY with three arguments, a Character nonvarying string, a Fixed Decimal constant 23, and a Bit string of '1'. The third will call VARY with the entry VARY as its one argument. NOTE: OPTIONS(VARIABLE) is intended for use with non-builtin subroutine that will except a variable number of arguments in its calling sequence like the PRIMOS entry point IOA$, in which the number of arguments and the format of them after the second is wholly dependent upon the format control items appearing in the first argument. It will however work in the global case as well. NOTE: beware of Machine restrictions apply to the case of NO arguments versus ANY arguments as the called subroutine may not execute properly under such circumstances.

7 Programmer Named Conditions

SPL allows the programmer to Create his own Condition names for use in an SPL program. The manners in which to do this is as follows: DECLARE <name> CONDITION; Then when the programmer wishes to reference that condition he uses the CONDITION builtin function as in: ON CONDITION(<name>) handle condition;, or as in: SIGNAL CONDITION(<name>);. These conditions are just as real as the PL1 and PRIMOS conditions are, and act in the same fashion.

8 -COPY, -QUICK, -MAP

SPL has three command line options to the compiler that may be of use to the programmer: -COPY/-NO_COPY, -QUICK/-NO_QUICK, and -MAP/-NO_MAP. SPL uses -COPY, -NO_QUICK, and -MAP as its defaults.

8.1 COPY/NO_COPY

The COPY/NO_COPY flag allows the programmer to suppress the copying of constants into temporary variables for subroutine calls. This feature must be coded properly in the called subroutine, for if it changes the value of one of its parameters which was passed as a constant, then the value of that constant in the calling program will be changed, causing subsequent use of that constant to use the wrong value. NO_COPY suppresses the copying of constants, COPY copies constants before calling subroutines. Use of the NO_COPY option can save on the amount of executable code generated by the SPL compiler.

8.2 QUICK/NO_QUICK

The QUICK/NO_QUICK flag allows SPL to call internal subroutines to the compiled program via a JSX instruction wherever possible. Internal subroutines that are recursive or are called from two different subroutines who are not a part of the same parents, cannot be Quick called and must ne invoked via a PCL. If NO_QUICK is specified, or the DEBUG option is specified, all internal subroutine are invoked via the PCL instruction. Local variables for subroutines invoked via the Quick mechanism are stored with the local variables of the first non-quicked parent of the called subroutine. The -QUICK command line option is the SPL replacement for the PLP procedure statement option: OPTIONS(SHORTCALL). The PLP option can be specified PER PROCEDURE. The SPL option encompasses all of the internal procedures of a compilation module. NOTE: the external procedures of a compilation module can NOT be Quick-called and must be PCLed.

8.3 MAP/-NO_MAP

The MAP option cantrols whether or not an identifier cross-reference map will appear in the listing file after the program listing. When a listing is specified, the MAP is the default. Use of -NO_MAP will compress the listing file to contain only the program listing. -XREF will force both the LISTING and MAP options on.

9 PLP features not supported

At rev 19.0, SPL does not support the following PLP BIFs: STACKPTR, LINKPTR, STACKBASE, ADDREL, BASEREL, BASEPTR, PTR, SEGNO, REL, RING, CSTORE nor does it support the PV REGFILE. SPL does not support the following procedure options: GATE, STACKROOT, or NOCOPY. These will all be implemented at a later rev. SPL does not support the use of the SHORTCALL option at 19.0. The functionality can be gotten through the use of the -QUICK command line option on the SPL command line. This feature can be overridden by the use ofthe option NONQUICK. Under the -QUICK flag, SPL will call via JSX only those routines which can be shortcalled, all others will be called via PCL. Use OPTIONS(NONQUICK) to ensure that all calls to the marked procedure are made through PCLs.

10 Bugs Fixed

The following bugs/problems were fixed for SPL rev 19.0.M2.

1. Use of '1'B or '0'B as constants in logical expressions of IF statements will no longer cause the OPTIMIZER to fail with ACCESS_VIOLATION$.

2. -Quick will no longer cause a random symbol table entry to be trashed causing other random errors to be generated.

3. Use of BIFs in SELECT statements will now work correctly instead of trying to do illegal conversions.

4. ENTRY OPTIONS(VARIABLE) will now work when the argument is a function call.

5. SPL will now tell you the correct number of errors if it has trouble with

11 Other changes to 19.0.M2

Here are the other changes to SPL for 19.0.M2

1. SPL will now generate 32 character external names. SEG will truncate these names to 8 characters for its own usage. BIND will accept all 32 characters.

2. -FIND_NODE command line option is now obsolete and unrecognized. The code generator has a better FIND_NODE routine that no longer requires the use of the command line option.

3. OPTIONS(CONSTANT) has been added as a means of placeing constant static data into the PROCEDURE frame instead of the LINKAGE frame of a program. This will allow the constants to be shared among many users if the program is shared. NOTE: Variables or arrays using this option should NOT be modified during the course of program execution. Attempts to do so will cause runtime errors from both shared programs and EPFs.

4. -32I command line option has been removed. SPL does not support generation of 32I mode code.

5. SPL will not allocate space for DATA ITEMS that are not used by the current program module. This includes the suppression of IPs to externals not referenced. It will however always generate COMMON BLOCK DEFINITION groups for all EXTERNAL STATIC defined in the program. The use of the -DEBUG command line option will force allocation of all declared names and IPs. Any external that is initialized by this program can be considered to be referenced.