Appendix A: Apply BNF

The following is a complete definition of the Apply language in the Extended Backus-Naur form. Square brackets enclose optional items, braces enclose items that may be repeated zero or more times, and vertical bars separate alternatives. These special symbols do not have their special meaning when they are enclosed in single quotes.

procedure ::= **PROCEDURE** *name (procedure-args)*
IS *variable-declarations*
BEGIN *statements*
END *name*;

name ::= *letter* { *letgit* | *_letgit* }

letgit ::= *letter* | *digit*

letter ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v
|w|x|y|z|A|B|C|D|E|F|G|H|I|J|K|L|M|N|O
|P|Q|R|S|T|U|V|W|X|Y|Z

digit ::= 1|2|3|4|5|6|7|8|9|0

procedure-args ::= *procedure-argument* { , *procedure-argument* }

procedure-argument ::= *var-list* : **IN** *type* [**BORDER** *border-type*]
[**SAMPLE** (*number* [, *number*])]
| *var-list* : **OUT** *type*
| *var-list* : **CONST** *type*

border-type ::= *number* | **MIRRORED** | **REPLICATED**

type ::= **ARRAY** (*range* [, *range*]) **OF** *elementary-type*
| *elementary-type*

range ::= *number* .. *number*

elementary-type ::= *sign-type* **BYTE** | *sign-type* **INTEGER** |
| **SHORT INTEGER** | *sign-type* **SHORT INTEGER**
| **SHORT** *sign-type* **INTEGER** | **REAL**
| **DOUBLE**

sign-type ::= [**SIGNED**] | **UNSIGNED**

number ::= *sign digit* { *digit* } | *sign* . *digit* { *digit* }
| *sign digit* { *digit* } . | *sign digit* { *digit* } . *digit* { *digit* }

sign ::= [–]

variable-declarations ::= { *var-list* : *type* ; }

var-list ::= *name* { , *name* }

bool-expr ::= *bool-expr* **AND** *bool-expr*
| *bool-expr* **OR** *bool-expr*
| **NOT** *bool-expr*
| (*bool-expr*)
| *expr b-op expr*

expr ::= *expr op expr*
| (*expr*)
| *name* | *name* (*subscript-list*)
| *number*
| *unary-op expr*
| **ABS**(*expr*) | **MAX**(*expr*) | **MIN**(*expr*)
| **INTEGER**(*expr*) | **FABS**(*expr*) | **SQRT**(*expr*)
| **LOG**(*expr*) | **LOG10**(*expr*) | **EXP**(*expr*)
| **SIN**(*expr*) | **COS**(*expr*) | **TAN**(*expr*)
| **ASIN**(*expr*) | **ACOS**(*expr*) | **ATAN**(*expr*)
| **ATAN2**(*expr*) | **REAL**(*expr*) | **DOUBLE**(*expr*)
| **ROW** | **COL**

unary-op ::= ! | –

subscript-list ::= *expr* { , *expr* }

b-op ::= < | <= | = | > | >= | /=

op ::= + | – | * | / | ^ | & | >> | '|' | <<

statements ::= stmt {stmt }

stmt ::= *assignment-stmt* | *if-stmt* | *for-stmt* | *while-stmt*

assignment-stmt ::= *name* := *expr;* | *name* (*subscript-list*) := *expr;*

if-stmt ::= **IF** *bool-expr* **THEN** *statements*
[ELSE *statements*] **END IF**;

for-stmt ::= **FOR** *name* **IN** *range* **LOOP** *statements* **END LOOP**;

while-stmt ::= **WHILE** *bool-expr* **LOOP** *statements* **END LOOP**:

NOTES

1. The precedence and associativity relationships for Apply operators are the same as precedence and associativity relationships in the C programming language with one exception. In Apply, bitwise and logical NOT have the same precedence.

2. The following reserved keywords must not be used in Apply routines:

   distributed
   local
   parallel
   copy

   These keywords are part of Apply's intermediate language. Appendix B describes the intermediate language.