SYSTEM HOUSE

A Description of the

Programming Language

MASCOT — MORAL

SOFTWARE SCIENCES

A Description of the

Programming Language

MASCOT ← MORAL

Reference No: 22/1132

Date: 16th February 1976

This is a first Draft Specification. Inevitably there will be detail changes during the development of the first translator and its integration into the MASCOT System.

Software Sciences Limited

Abbey House, Farnborough Road, Farnborough, Hampshire

Telephone: Farnborough 44321

Telex: 858228. Cables: Softwares Farnborough Hampshire

Preface

MORAL is a programming language designed to integrate with the MASCOT System. The name stands for 'MASCOT-oriented Reliable Application's Language'.

The language was designed by the author during the period November 1975 to January 1976 under Contract for the Royal Radar Establishment at Malvern in England.

Two influences have had a pronouned effect upon the language. Because it is designed specifically to support the development of real-time Systems using MASCOT, MORAL contains no mechanism for communication with any environment other than that of a MASCOT Activity. Secondly, MORAL is designed specifically to be implemented initially via translation into CORAL 66 and this has constrained the language in a number of ways which may seem surprising in the context of current language design practice.

H.F.Harte
Software Sciences Ltd
February 1976.

CONTENTS

1. INTRODUCTION

This description takes the form of a conventional syntax accompanied by prose explanations of the semantics of each construction.

The notation adopted is basically that used in the official definition of CORAL 66. Class names are written as single words, beginning with a capital letter but otherwise in lower case.

The classname being defined is followed by an equals sign and then one or more alternative forms in which the class can be written. Where alternatives are separated by explanatory text and/or other rules the classname being defined is repeated for subsequent alternatives.

Each alternative of every rule is numbered at the right in the form R.A where R is the rule-number and A the alternative-number. For a rule with only one alternative only the rule-number is used.

Following the main description, Section 3 contains the collected syntax rules in alphabetical order with their rule number to facilitate finding them in the main text.

2. THE MORAL LANGUAGE

2.1 Data

2.1.1 Simple Data Types

In MORAL, information is represented and manipulated in the form of the values taken by data variables of many types. The basic types from which all other types are derived are the Simpletypes

```
Simpletype  =  'INTEGER'              1.1
```

An 'INTEGER' variable takes values which are the positive and negative whole numbers within some range determined by the size and form of a machine word in the execution environment.

```
Simpletype  =  'INTEGER' (Range)      1.2
```

An 'INTEGER' (Range) variable takes values within its explicitly specified range

```
Range  =  Bound 'TO' Bound            2.1
          Bound  :   Bound            2.2
Bound  =  Integerconstant             3
```

The use of a colon is purely a lexical alternative in a range. The value of the second Bound must be greater than or equal to the value of the first Bound. Both must lie within the range of a normal 'INTEGER' variable.

```
Simpletype  =  'FIXED' (Size, Fractionbits)
                                      1.3
```

A 'FIXED' (Size, Fractionbits) Variable takes values which are representable as a fixed-point binary number within a total of 'Size' bits including sign.'Fractionbits' gives the number of bits of the representation, counting from the least significant end, which are to be regarded as falling after the binary point.

Size = Integerconstant 4

Size must be positive and not greater than the number of bits available in an execution environment word.

Fractionbits = Integerconstant 5

Fractionbits may be positive or negative. Where the apparent position of the binary point falls outside the stored representation of the value, zeros are assumed to occupy the positions between the point and the stored portion of the number (except in the case of positions at the most significant end where the virtual bits are assumed to be such as to preserve the sign.)

Simpletype = 'FLOATING' 1.4

A 'FLOATING' Variable takes values which are those provided by the single-precision floating-point facilities within the execution environment.

Simpletype = 'BYTE' 1.5

A 'BYTE' Variable takes values which are the positive integer interpretations of the bit-patterns comprising the character set of the execution environment.

Simpletype = 'STATUS' (Identifierlist) 1.6

A 'STATUS' (Identifierlist) Variable takes values as listed in the Identifierlist. No numerical interpretation is implied although the order of the Identifiers is significant in defining the relationships of lessthan and greaterthan.

> <u>example</u>: A 'STATUS' (RED, REDANDAMBER, GREEN, AMBER) takes the four values RED, REDANDAMBER, GREEN and AMBER.
>
> <u>note</u>: A status value such as RED only has a meaning in a context which defines a status type including RED as one of its values. The same identifier may be used in several Status types without confusion.
> The same identifier may <u>not</u> be used to declare a Variable of a Status type including itself among its values.

Simpletype = Usertype 1.7

Syntactically a Usertype is also treated as a simple type. A Usertype is a type which has been given a specific user-defined name in a type definition (Typedec). It is written in quotes as for a built-in type.

Usertype = 'Identifier' 6

A Usertype Variable takes values appropriate to its definition.

2.1.2 Arrays

```
Arraytype    =   'ARRAY' (Range) Simpletype        7.1
                 'ARRAY' (Range,Range)Simpletype   7.2
```

An Arraytype Variable is a conventional one or two-dimensional array of Simpletype Variables and its value is the composite set of the values of its Simpletype elements. Each subscript range is as defined for a ranged -integer type.

Arraytypes and Simpletypes are together classified as Directtypes.

```
Directtypes   =   Arraytype                        8.1
                  Simpletype                       8.2
```

2.1.3 References

```
Referencetype   =   'REF' Directype                9.1
```

A Referencetype Variable takes values which are references to variables of the specified Directtype.

Referencetypes and Directtypes are together classified as Assignabletypes.

```
Assignabletype   =   Referencetype                10.1
                     Directtype                   10.2
```

2.1.4 Constant Variables

A Constanttype Variable may have a value of some specified Assignabletype, but it may not have a new value assigned to it. It remains constant throughout its existence.

```
Constanttype   =   'CONST' Assignabletype        11
```

References to Constanttype Variables are also describable.

```
Referencetype   =   'REF' 'CONST' Directtype    9.2
```

The limitation to Directtype in this last case is so that types may always be described fairly briefly. If the effect of a " 'REF' 'CONST' 'REF' Sometype" is required this can be achieved by introducing a named Usertype defined to be 'REF' Sometype.

Constanttypes and Assignabletypes are together classified as Declarabletypes.

```
Declarabletype   =   Constanttype              12.1
                     Assignabletypes           12.2
```

2.1.5 Structured Data

Compound data types whose values are aggregates of field values are provided in MORAL in the form of Group data types. These are described in Section 2.5.

2.2 Declaring Variables

A Variable is introduced into a program by the appearance of a Datadec.

```
Datadec  =  Declarabletype Kpidlist           13

Kpidlist = Identifier Keyoption Presetoption 14.1
           Identifier Keyoption Presetoption,Kpidlist
                                             14.2
```

The meaning of Keyoption when not empty is explained in Section 2.6.1.1

Each identifier which appears in the Kpidlist is the name by which a Variable of the specified Declarabletype is to be known. The Variable exists and may be referred to only within the Block (rule 68) in which its declaration occurs. The same identifier may not be used for more than one Variable within the same Decs (rule 69). Where the same identifier is used to name a further Variable declared within an inner Block then occurrences of the identifier are taken to refer to that further variable within the inner Block.

The Presetoption is used to give a variable an initial value and takes the form

Presetoption	=	:= Presetunit	15.1
		Empty	15.2

In the case of a declaration of a Simpletype Variable the Presetunit must be a single constant of the correct type.

Presetunit	=	Constant	16.1

For declarations of compound data variables (arrays and Usertypes which have been defined as Groups) the Presetunit must, if present, supply the correct number and type of constants corresponding to the basic field and element variables of the compound. However, where it is inappropriate or impossible to supply a particular field with a preset value the corresponding constant may be omitted.

Presetunit	=	(Presetsequence)	16.2
		Empty	16.3

Presetsequence = Presetunit 17.1

Presetunit, Presetsequence 17.2

2.3 Constant Values

The forms of a Constant Value as they may appear in a Presetoption are the following

Constant = Addoperator Unsignedconstant 18.1

Unsignedconstant 18.2

Numeric constants may be signed

Addoperator = + 19.1

\- 19.2

Unsignedconstant = Number 20.1

Identifier 20.2

In the case of an Identifier, this must be one which has already been declared to stand for a constant variable of the required type and to have been given a preset value. Syntactically a status-value is also included in this case.

Number = Real 21.1

Integer 21.2

Real = Digitlist. Digitlist 22.1

Digitlist$_{10}$ Signedinteger 22.2

Digitlist. Digitlist$_{10}$ Signedinteger 22.3

$_{10}$Signedinteger 22.4

Signedinteger = Addoperator Integer 23.1

Integer 23.2

Integer = Digitlist		24
Digitlist = Digit		25.1
	Digit Digitlist	25.2
Digit = 0/1/2/3/4/5/6/7/8/9		26

For 'BYTE' Constants either the integer form may be used (non-negative) or a character form. The character form also extends as a shorthand for arrays of Bytes

Constant = String		18.3
String = "Stringitemlist"		27
Stringitemlist = Stringitem Stringitemlist		28.1
	Stringitem	28.2
Stringitem = Anycharacterotherthanquotationmarks		
	" "	29.1

A pair of quotation symbols are used to stand for a single quotation symbol within a string. A string is regarded as equivalent to one constant value for each character within it.

One other form of constant is available.

Constant = 'NIL'	18.4

'NIL' can be used for any Referencetype. A Reference variable with this value refers to no target variable and may be regarded as being 'zero'.

2.4 Usertype Declarations

A Usertype is introduced by a Typedec.

Typedec = 'TYPE' Usertype = Typedefiner	30

```
Typedefiner = Declarabletype                    31.1
              Proctype                          31.2
              Groupdefinition                   31.3
```

For Proctype see 2.9 . We can now describe the form of a Groupdefinition.

2.5 Groups

```
Groupdefinition = Denseoption 'GROUP' Fields 'ENDGROUP'
                                                  32
```

A Group-type is defined in terms of its Fields. The Denseoption is a qualification which relates to the way in which the individual Fields are mapped onto the storage used in the execution environment.

```
Denseoption ='DENSE'                            33.1
             Empty                              33.2
```

A description of the storage layout for a 'DENSE' Group is a question of implementation and falls outside the scope of the present document.
The Fields of a Group are defined in the same way that variables are declared with certain additional features.

```
Fields = Fieldsunit ; Fields                    34.1
         Fieldsunit                             34.2

Fieldsunit = Fielddec                           35.1
             Lockedsequence                     35.2
```

A Lockedsequence defines fields which may only be selected from variables of this grouptype if an acceptable key has been associated with the variable.

```
Lockedsequence = Lockedset Lockedsequence     36.1
                 'UNLOCK'                     36.2

Lockedset = 'LOCK' Keys Readonlyoption
                        Fieldsequence         37
```

Each lockedset specifies a number of keys which may be used to give access to the Fieldsequence contained within it.

```
Keys = Identifierlist                         38.1
       Empty                                  38.2
Identifierlist = Identifier                   39.1
                 Identifier, Identifierlist   39.2
```

Each of these identifiers denotes a key which will give full access to the Fieldsequence of the Lockedset. Keys may also be defined which give Read-only access

```
Readonlyoption = 'READONLY' Identifierlist    40.1
                 'READONLY' 'OPEN'            40.2
                 Empty                        40.3
```

The use of 'OPEN' specifies that no key is required to Read the fields of the Fieldsequence.

```
Fieldsequence = Fielddec                      41.1
                Fielddec ; Fieldsequence      41.2
```

A Fieldsequence is a series of individual Fielddecs. A Fielddec is either the declaration of a single field variable or may define a set of alternative layouts.

```
Fielddec = Datadec                            42.1
           Proceduredec                       42.2
```

Procedures may be defined as if they were fields of the grouptype.

```
Fieldec = Casedec                                   42.3
```

This is the form for alternative layouts.

```
Casedec = 'CASE' Identifier Casedecwhenset
                            'ENDCASE'            43
```

The identifier must name a field variable of either 'INTEGER', 'INTEGER' (Range) or 'STATUS' (identifierlist) type which has already been declared as part of this Groupdefinition (and not within a separate arm of an enclosing Casedec). The following field structure is then defined according to the value of the nominated field variable.

```
Casedecwhenset = Casedecother                       44.1
                 Casedecwhen                        44.2
                 Casedecwhen Casedecwhenset         44.3
```

Each Casedecwhen gives the fieldsequence corresponding to certain of the values which may be taken by the controlling field.

```
Casedecwhen = 'WHEN'  Cases : Fieldsequence         45
Cases = Caseunit                                    46.1
        Caseunit, Cases                             46.2

Caseunit = Constant                                 47.1
           Constant 'TO' Constant                   47.2
```

The constant must correspond to the type of the controlling field variable.

The final arm of the casedec may take the form

```
Casedecother = 'WHEN' 'OTHER' : Fieldsequence       48
```

This specifies the fieldsequence which is to correspond to all the possible values of the controlling field variable which have not already been used in a case-arm.

The values specified for different arms must not have any common values.

2.6 Expressions

An expression is the construction by which new values may be created through the application of operators upon existing values.

2.6.1 Primaries

2.6.1.1 Variables

A Primary is the basic form of operand

Primary = Variable 49.1

The most important Primary is the use of a declared Variable

Variable = Identifier 50.1

Syntactically this embraces two cases:- Direct access by name to a Variable's current value or in a reference context to the name itself and invocation of a procedure which requires no parameter and returns a value;

Variable = Variable. Identifier 50.2

This form denotes selection of a field-variable 'Identifier' from a Variable of a Group type. It is subject to restriction in the case of fields which have been defined within Locked sets. For such a case the Variable from which the field is being selected must be associated with an acceptable Key in the context of use of the selected field. Association of a key with a variable is established by the appearance of the key-identifier in a Keyoption in the Declaration of the variable (or in the case of a parameter of a procedure in the Parameter Specification).

Keyoption = (Identifier) 51.1
Empty 51.2

The third form of a Variable represents either access to an element of an array or a call to a procedure requiring parameters which delivers a result.

Variable = Variable (Parameterpack) 50.3

In the case of an Array the Parameterpack must supply the appropriate integer-valued Subscripts, both in number and range. In the case of a procedure the Parameter pack must supply values matching the procedure. The final form of a Variable represents access via a reference variable to the further variable to which it currently refers.

Variable = [Variable] 50.4

The Variable must be of a Referencetype.

2.6.1.2. Values

The second form of Primary is direct reference to a value

Primary = Value 49.2

```
Value = Number                                    52.1
        String                                    52.2
        'NIL'                                     52.3
```

These cases are described in Section 2.3

```
Value = Identifier                                52.4
```

The identifier here must be a status value and can only be used in a situation where the statustype involved is defined by the context in which the value appears. The contexts which satisfy this criterion are the right-hand sides of assignment statements and comparisons and components of a Parameterpack.

```
Value = Display                                   52.5
```

This final form of value applies to compound values of either array or group type. There are two forms, each with two variations.

```
Display = 'PRESET' (Presetsequence)               53.1
```

A 'PRESET' Display must contain only values which are constant and is analagous to the presetting of a variable when it is declared. This form may only appear in a context which defines its type. A second preset form is available where the context does not define its type:-

```
Display = 'PRESET' Assignabletype : (Presetsequence)
                                                  53.2
```

There are also two forms of display in which the elements are dynamically evaluated each time the display is used. Again the second of these specifies its type explicitly.

```
Display = 'EVAL' (Parameterpack)                    53.3
          'EVAL' Assignabletype :(Parameterpack)
                                                    53.4

Parameterpack = Expression                          54.1
                Expression, Parameterpack           54.2
```

2.6.1.3 Other Primaries

There are three further types of Primary

A nested expression:-

```
Primary = (Expression)                              49.3
```

A typed expression:-

```
Primary = Assignabletype :(Expression)              49.4
```

Access to a section of the Binary representation of a Variable:-

```
Primary = 'BITS' (Integerconstant, Integerconstant)
                 'OF' Variable                      49.5
```

The first Integerconstant specifies the number of bits to be accessed; the second specifies the bit-position of the least significant of the bits accessed taking zero as the position of the least-significant bit of a variable.

The value involved is the positive integer interpretation of the bits accessed.

2.6.2 Operations

```
Simpleexpression = Term                                55.1
                   Addoperator Term                    55.2
                   Simpleexpression Addoperator Term
                                                       55.3

Term = Factor                                          56.1
       Term Multoperator Factor                        56.2

Factor = Logicalterm                                   57.1
         Factor 'DIFFER' Logicalterm                   57.2

Logicalterm = Logicalfactor                            58.1
              Logicalterm 'UNION' Logicalfactor
                                                       58.2

Logicalfactor = Primary                                59.1
                Logicalfactor 'MASK' Primary           59.2
```

The operators 'DIFFER', 'UNION' and 'MASK' are the logical operators Exclusive-or, Inclusive-or and Logical-and.

```
Multoperator = *                                       60.1
               /                                       60.2
```

All the above operators apply to numeric operands of types 'INTEGER', 'INTEGER' (Range), 'FIXED' (Size, Fractionbits) and 'BYTE'. Addoperator and Multoperators also apply to 'FLOATING' operands.

Operands of Addoperators are always converted to the same type before the Addoperator is applied. The type to which they are converted is either a) the type of the left operand if the context is weak or b) the type required by the context if the context is strong.

An Expression is in a strong context if it is in the right hand side of an assignment or comparison or if it is in a Parameterpack or if it is in a statement form requiring an Integer value.

'BYTE' operands are always converted to 'INTEGER' as a first step before further conversion or before direct application of an operator. This is the only conversion ever applied for operands of the Logical operators.

Usertypes which have been defined to be acceptable operand types are convertible to their defined base-type for the purpose of operations. However, only one derived Usertype is allowed among the Terms of a Simpleexpression.

(For example:- If we have 'TYPE' 'AGE' = 'INTEGER' and 'TYPE' 'SIZE' = 'INTEGER' and we declare 'AGE' A1, A2; 'SIZE' S1,S2; then A1+A2, A1+10, S1+S2, S1+8 all are legal but A1+S1 is illegal. Notice that A1+S1*1 <u>is</u> legal since the result of an operation is always taken as the base type).

2.6.3. Conditional Expressions

```
Expression = Simpleexpression                              61.1
             'IF'Condition 'THEN' Expression 'ELSE'
                  Expression 'FI'                          61.2
```

2.7 Conditions

```
Condition = Subcondition                                   62.1
            Condition 'OR' Subcondition                    62.2
```

Conditions are evaluated only as far as the first true Subcondition if any.

```
Subcondition = Conditionelement                     63.1
               Subcondition 'AND' Conditionelement
                                                    63.2
```

Subconditions are evaluated only as far as the first false conditionelement if any.

```
Conditionelement = Comparison                       64.1
                   Variable                         64.2
```

A Variable is a sufficient conditionelement only if it is a statustype including TRUE among its status values.

```
Comparison = Expression Comparator Expression
                                                    65
```

The result of the second expression must be convertible into the type of the first expression.

```
Comparator = < | <= | = | >= | > | <>               66
```

<> means not-equal-to. Comparisons are permissable between Expressions of any type. However only = and <> may be used between Expressions of types other than the numeric types and the status types.

2.8 Statements

2.8.1. Blocks

```
Statement = Block                                   67.1
```

A statement may be a nested Block

```
Block = 'BEGIN' Decs Statementsequence 'END' 68
Decs = Dec ; Decs                            69.1
       Empty                                 69.2
Dec = Datadec                                70.1
      Proceduredec                           70.2
      Typedec                                70.3
Statementsequence = Statement                71.1
                    Statement ; Statementsequence
                                             71.2
```

2.8.2 Assignmentstatements

```
Statement = Assignementstatement             67.2
```

The form of an Assignmentstatement is conventional

```
Assignmentstatement = Variable := Expression 72
```

The Expression must result in a value of a type convertible into that of the Variable.

2.8.3. Conditionalstatements

```
Statement = Conditionalstatement             67.3

Conditonalstatement = 'IF' Condition 'THEN'
                      Statementsequence Elseoption
                      'FI'                   73

Elseoption = 'ELSE' Statementsequence        74.1
             Empty                           74.2
```

2.8.4. Loopstatements

```
Statement= Loopstatement                     67.4

Loopstatement = Controlpart Body Tail        75
```

The Body of a Loopstatement is executed repeatedly under the control of the Controlpart. The Tail is executed once, following the last execution of the body according to the controlpart. If an Escapestatement is obeyed inside the Body the Loopstatement terminates immediately without execution of the Tail.

```
Body = 'DO' Statementsequence                    76
```

The forms of the Controlpart are described below:-

```
Controlpart = 'FOR' Controlspec                  77.1
```

The controlspec defines a control-variable and set of values which it will take in turn, the Body being executed in the context of each of these values. Within the Body the Control-variable is available as though it had been declared to be'CONST' preset to the present value. i.e. it may not be assigned to.

Four forms of Controlspec are available

```
Controlspec = Rangetype Identifier               78.1
Rangetype = 'INTEGER' (Range)                    79.1
            'STATUS' (Identifierlist)            79.2
            Usertype                             79.3
```

The identifier stands for the Control-variable which takes the values in the range of the Rangetype. The Usertype here must be defined to be of a suitable Rangetype.

```
Controlspec = Declarabletype Identifier 'FROM' Primary
                                                 78.2
```

In this form of Controlspec the Primary must be of the type 'ARRAY'(....) Declarabletype. The Controlvariable is successively given the values of the elements of the array.

Controlspec =	Pointertype Identifier 'OVER' Primary	78.3
Pointertype =	Referencetype	80.1
	Usertype	80.2

This form is similar but now the Control-variable is of a referencetype (or Usertype defined as such) and successively takes values which refer to the elements of the array, which must be of a suitable type.

Controlspec =	Identifier Fromoption Byoption Tooption	78.4

This form is the conventional loop control with an 'INTEGER' Controlvariable stepping through the set of values specified by the From, By and Tooptions.

Fromoption =	'FROM' Expression	81.1
	Empty	81.2

The Expression must provide an Integer starting value. If Empty the value zero is used.

Byoption =	'BY' Expression	82.1
	Empty	82.2

The expression must provide an Integer step-value by which the control-variable is incremented at the end of each execution of the Body.

If Empty the value 1 is used. The expression is evaluated once only at the start of the Loopstatement and the resulting value used thereafter.

```
Tooption = 'TO' Expression                    83.1
           Empty                              83.2
```

The Expression must provide an Integer value which defines the end of the loop. The Expression is evaluated once at the start of the loop. If Empty the largest available integer value is used. The Loopstatement is complete where the last execution of the Body is followed by an increment which gives the control-variable a value greater than the limit given by the Tooption.

The second form of a Controlpart is:-

```
Controlpart = 'WHILE' Condition               77.2
```

In this case the number of executions of the Body is not specified. It will continue to be executed repeatedly as long as the Condition is satisfied.

```
Controlpart = Empty                           77.3
```

If the Controlpart is Empty the Body is executed repeatedly. The only termination available in this case is either an 'ESCAPE' or a direct jump out of the loop. The Tail will never be executed in this form of Loopstatement.

```
Tail = 'THEN' Statementsequence 'ENDLOOP'     84.1
       'ENDLOOP'                              84.2
```

The Statementsequence in the Tail is executed when the Loopstatement terminates <u>according to the Controlpart</u>.

2.8.5. Casestatements

Statement = Casestatement 67.5

Casestatement= 'CASE' Expression Whenset 'ENDCASE' 85

The whenset consists of a number of arms, one of which will be executed according to the value of the Expression. The Expression must result on a value of type 'INTEGER', 'INTEGER (Range), 'BYTE' or 'STATUS' (Identifierlist)

```
Whenset = Lastwhen                               86.1
          Whenelement                            86.2
          Whenlement Whenset                     86.3

Whenelement = 'WHEN' Cases : Statementsequence
                                                 87
Cases = Caseunit                                 88.1
        Caseunit, Cases                          88.2
Caseunit = Constant                              89.1
           Constant 'TO' Constant                89.2
```

The Statementsequence of the Whenelement is chosen for execution if and only if the value resulting from the Case Expression is included within the Cases either directly or within the range of a Constant 'TO' Constant.

```
Lastwhen = 'CASE' 'OTHER' : Statementsequence
                                                 90
```

The Last arm of a case statement may take this form. The Statementsequence of a Lastwhen is chosen for execution if and only if the value resulting from the case expression is not included within the Cases of any Whenelement of the Casestatement.

No value may be included within the cases of more than one Whenelement.

2.8.6 Procedurestatements

Statement = Procedurestatement	67.6
Procedurestatement = Variable	91

Syntactically the Parameterpack is part of the Variable. This statement is a call to the Procedure involved. A suitable Parameterpack to match the Procedure's Parameterspeclist must be supplied.

2.8.7 Dummystatements

Statement = Dummystatement	67.7
Dummystatement = Empty	92

The execution of a Dummystatement naturally produces no effect whatsoever.

2.8.8 Labels

Statement = Label : Statement	67.8
Label = Identifier	93

Any statement may be labelled provided the identifier used does not clash with that of a variable or another label declared within the smallest enclosing Block.

2.8.9 Jumps

Statement = 'GOTO' Label	67.9

Execution of this statement causes immediate transfer of control to the Statement labelled with the given Label and occurring within an enclosing Block. Normal scope rules apply in that identifiers within inner Blocks supersede identifiers with the same spelling in outer Blocks for the duration of the Block containing them.

2.8.10 Structure-related Jumps

Statement = 'ANSWER' Expression 67.10

This statement may only appear within a procedure which delivers a value. It causes immediate termination of the execution of the Procedure with the value resulting from the Expression being returned as the result of the Procedure.

Statement = 'RETURN' 67.11

This statement may only appear within a Procedure which does not return a value and it causes immediate termination of the execution of the Procedure.

Statement = 'REPEAT' 67.12

This Statement may only appear within the Body of a Loopstatement and causes the execution of the current iteration to be terminated. The Controlpart then determines whether there are to be further iterations in the normal way.

Statement = 'ESCAPE' 67.13

This Statement may only appear within the Body of a Loopstatement and causes the execution of the entire Loopstatement to be terminated without execution of the Tail.

2.8.11 Mascotstatements

Statement = Mascotstatement 67.14

A number of Statement forms are provided to make the Mascot Primitive-operations available. For a fuller explanation of the meaning of these operations the Reader is referred to the Mascot System.

Mascotstatement = 'JOIN' Variable 94.1

The Variable must provide a variable of type 'CONTROLQ', a Usertype predefined in MORAL. This statement requests control of the Control-queue in question and the Activity making the request will be suspended until Control can be given.

Mascotstatement = 'WAIT' Variable 94.2

Again the Variable is of type 'CONTROLQ'. The Statement causes the executing Activity to be suspended until a 'STIM' is executed by another Activity upon the same Control-queue. The fact that a 'STIM' has occurred is remembered, and if one has already been performed the activity proceeds immediately and the memory of the 'STIM' is cancelled.

'WAIT' and 'LEAVE' may only be executed by an Activity which has control of the Control-queue in question.

Mascotstatement = 'STIM' Variable 94.3

This statement gives a 'STIM' to the Control-queue nominated.

Mascotstatement = 'LEAVE' Variable 94.4

This statement releases control of the Control-queue.

Mascotstatement = 'DELAY' Expression 94.5

An activity executing this statement is suspended and does not become eligible for scheduling again for the number of time units given by the value of the Expression.

Mascotstatement =	'JOININT' Variable	94.6
	'WAITINT' Variable	94.7
	'LEAVEINT'Variable	94.8
	'SETTRANSFER' Primary, Variable	94.9

These four Statement forms apply to Variables of type 'INTERRUPT', a second predefined Usertype.'JOININT' and 'LEAVEINT' respectively request and release control of the Interrupt-Control-queue, and'WAITINT' requests a transfer on the associated Interrupt-driven Device, the Activity being suspended while this transfer occurs. A SETTRANSFER Statement establishes the data-address associated with the transfer. The Primary gives that address in the form of a Referencetype value. The Variable identifies the Interrupt Control-queue in the normal way.

'WAITINT', 'LEAVEINT' and SETTRANSFER' may only be executed by an activity having control of the Interrupt Control-queue involved.

Mascotstatement = 'SUSPEND' Integerconstant 94.10

This statement causes the Activity executing it to be suspended but it immediately becomes reavailable for scheduling at the back of the Scheduler's list with priority given by the Integerconstant.

2.9 Procedures

```
Proceduredec = Answerspecoption Recursivity
                    Prochead ; Statement    95

Answerspecoption = Answerspec               96.1
                   Empty                    96.2
Answerspec = Referencetype                  97.1
             Simpletype                     97.2
```

The Answerspec gives the type of value if any returned by the procedure. Array and Group values cannot be returned except by reference.

```
Recursivity = 'RECURSIVE'                   98.1
              'PROCEDURE'                   98.2
```

Only procedures declared as 'RECURSIVE' may be called recursively.

```
Prochead = Identifier                       99.1
           Identifier (Parameterspeclist)   99.2
```

The Prochead gives the name of the Procedure and declares its formal parameters if any.

```
Parameterspeclist = Parametergroup          100.1
                    Parametergroup ; Parameterspeclist
                                            100.2
Parametergroup = Paramtype Kidlist          101
```

Paramtype = Answerspec 102.1
'CONST' Answerspec 102.2
Proctype 102.3

Parameter types are restricted to these types, Proctype is not strictly a data type but procedures as parameters are supported.

The Parameter mechanism in MORAL is basically 'call-by -value' although the value involved may be a reference-value and give access to actual parameter data. It is as though each formal parameter is assigned its value from the actual parameter list and is thereafter an independent variable in its own right.

In the case of 'CONST' parameters this assignment is a dynamic presetting operation.

In a parameterspeclist one parameter of a type involving 'REF' 'ARRAY' (....) may use as its Bounds in array subscript ranges another parameter from earlier in the paramspeclist provided this is a 'CONST' 'INTEGER' parameter.

In this way it is permissable to communicate the size of an array in a parameter list.

Kidlist = Identifier Keyoption 103.1
Identifier Keyoption,Kidlist 103.2

Formal parameters may be given associated Keys in just the same way as normal variables.

```
Proctype = Answerspecoption Recursivity      104.1
           Answerspecoption Recursivity (Paramtype-
                                  speclist) 104.2
```

A parameter of type procedure is specified in a similar manner to the declaration of a procedure, except that any parameters of the parameter procedure are left unnamed and no statement is now necessary.

```
Paramtypespeclist = Paramtype                105.1
                    Paramtype, Paramtypespeclist
                                             105.2
```

2.10 Comments

Two forms of Comment may be inserted into a program text without affecting the execution of the program in any way.

The form:- 'COMMENT' any text not containing semicolon ; may be inserted anywhere where a Dec or a Statement would be legal.

Also the form ;(Any text in which parentheses are matched) may be inserted anywhere as an alternative to a semicolon by itself.

2.11 Macros

The MORAL translator includes a macro processor which processes the text before it is analysed as a MORAL text. This is in fact a standard CORAL 66 macro-processor.

A macro definition may be written anywhere where a declaration or statement could occur and takes one of the forms:-

'DEFINE' Identifier String ;

'DEFINE' Identifier (Identifierlist) String ;

A macro definition may be cancelled by the form:-

'DELETE' Identifier ;

A macro Identifier may also be redefined without deleting it.

Between the Definition of a macro and its deletion or redefinition occurrences of the macro identifier will be expanded into the Defined String. Actual parameters may be any strings of characters in which brackets are matched and any commas are protected by brackets. The number of actual parameters must match the number of formals.

The expanded text of a macro may contain both macro calls and macro definitions and will be processed only after the expansion of the basic macro has been performed.

Complete Syntax in Alphabetical Order

The numbers at the left are the rule numbers in the order in which they appear in the main specification.

```
19)   Addoperator           =   +
                                -
97)   Answerspec            =   Referencetype
                                Simpletype
96)   Answerspecoption      =   Answerspec
                                Empty
 7)   Arraytype             =   'ARRAY' (Range) Simpletype
                                'ARRAY' (Range, Range) Simpletype
10)   Assignabletype        =   Referencetype
                                Directtype
72)   Assignmentstatement   =   Variable := Expression
68)   Block                 =   'BEGIN' Decs Statementsequence 'END'
76)   Body                  =   'DO' Statementsequence
 3)   Bound                 =   Integerconstant
82)   Byoption              =   'BY' Expression
                                Empty
43)   Casedec               =   'CASE' Identifier Casedecwhenset
                                                      'ENDCASE'
48)   Casedecother          =   'WHEN' 'OTHER' : Fieldsequence
45)   Casedecwhen           =   'WHEN' Cases : Fieldsequence
44)   Casedecwhenset        =   Casedecother
                                Casedecwhen
                                Casedecwhen Casedecwhenset
46)   Cases                 =   Caseunit
                                Caseunit, Cases
85)   Casestatement         =   'CASE' Expression Whenset 'ENDCASE'
47)   Caseunit              =   Constant
                                Constant 'TO' Constant
66)   Comparator            =   = | < | > | <= | >= | <>
65)   Comparison            =   Expression Comparator Expression
```

```
62)   Condition                =   Subcondition
                                   Condition 'OR' Subcondition
73)   Conditionalstatement     =   'IF' Condition 'THEN'
                                        Statementsequence Elseoption'FI'
64)   Conditionelement         =   Comparison
                                   Variable
18)   Constant                 =   Addoperator Unsignedconstant
                                   Unsignedconstant
                                   String
                                   'NIL'
11)   Constantype              =   'CONST' Assignabletype
77)   Controlpart              =   'FOR' Controlspec
                                   'WHILE' Condition
                                   Empty
78)   Controlspec              =   Rangetype Identifier
                                   Declarabletype Identifier 'FROM'
                                                         Primary
                                   Pointertype Identifier 'OVER' Primary
                                   Identifier Fromoption Byoption
                                                         Tooption
13)   Datadec                  =   Declarabletype Kpidlist
70)   Dec                      =   Datadec
                                   Proceduredec
                                   Typedec
12)   Declarabletype           =   Constanttype
                                   Assignabletype
69)   Decs                     =   Dec ; Decs
                                   Empty
33)   Denseoption              =   'DENSE'
                                   Empty
26)   Digit                    =   0/1/2/3/4/5/6/7/8/9
25)   Digitlist                =   Digit
                                   Digit Digitlist
8)    Directtype               =   Arraytype
                                   Simpletype
```

```
53)   Display               =   'PRESET' (Presetsequence)
                                'PRESET' Assignabletype :
                                               (Presetsequence)
                                'EVAL' (Parameterpack)
                                'EVAL' Assignabletype :
                                               (Parameterpack)
92)   Dummystatement        =   Empty
74)   Elseoption            =   'ELSE' Statementsequence
                                Empty
61)   Expression            =   Simpleexpression
                                'IF' Condition 'THEN' Expression
                                              'ELSE' Expression 'FI'
57)   Factor                =   Logicalterm
                                Factor 'DIFFER' Logicalterm
42)   Fielddec              =   Datadec
                                Proceduredec
                                Casedec
34)   Fields                =   Fieldsunit ; Fields
                                Fieldsunit
41)   Fieldsequence         =   Fielddec
                                Fielddec ; Fieldsequence
35)   Fieldsunit            =   Fielddec
                                Lockedsequence
5)    Fractionbits          =   Integerconstant
81)   Fromoption            =   'FROM'Expression
                                Empty
32)   Groupdefinition       =   Denseoption 'GROUP' Fields 'ENDGROUP'
/)    Identifier            =   Letter
                                Identifier Letter
                                Identifier Digit
39)   Identifierlist        =   Identifier
                                Identifier, Identifierlist
24)   Integer               =   Digitlist
/)    Integerconstant       =   Addoperator Integer
                                Integer
                                Addoperator Identifier
                                Identifier
```

51)	Keyoption	=	(Identifier)
			Empty
38)	Keys	=	Identifierlist
			Empty
103)	Kidlist	=	Identifier Keyoption
			Identifier Keyoption, Kidlist
14)	Kpidlist	=	Identifier Keyoption Presetoption
			Identifier Keyoption Presetoption, Kpidlist
93)	Label	=	Identifier
90)	Lastwhen	=	'WHEN' 'OTHER' : Statementlist
/)	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
36)	Lockedsequence	=	Lockedset Lockedsequence
			'UNLOCK'
37)	Lockedset	=	'LOCK' Keys Readonlyoption Fieldsequence
59)	Logicalfactor	=	Primary
			Logicalfactor 'MASK' Primary
58)	Logicalterm	=	Logicalfactor
			Logicalterm 'UNION' Logicalfactor
75)	Loopstatement	=	Controlpart Body Tail
94)	Mascotstatement	=	'JOIN' Variable
			'WAIT' Variable
			'STIM' Variable
			'LEAVE' Variable
			'DELAY' Expression
			'JOININT' Variable
			'WAITINT' Variable
			'LEAVEINT' Variable
			'SETTRANSFER' Primary, Variable
			'SUSPEND' Integerconstant
60)	Multoperator	=	*
			/

21)	Number	=	Real
			Integer
101)	Parametergroup	=	Paramtype Kidlist
54)	Parameterpack	=	Expression
			Expression, Parameterpack
100)	Parameterspeclist	=	Parametergroup
			Parametergroup ; Parameterspeclist
102)	Paramtype	=	Answerspec
			'CONST' Answerspec
			Proctype
105)	Paramtypespeclist	=	Paramtype
			Paramtype, Paramtypespeclist
80)	Pointertype	=	Referencetype
			Usertype
15)	Presetoption	=	:= Presetunit
			Empty
17)	Presetsequence	=	Presetunit
			Presetunit, Presetsequence
16)	Presetunit	=	Constant
			(Presetsequence)
			Empty
49)	Primary	=	Variable
			Value
			(Expression)
			Assignabletype : (Expression)
			'BITS' (Integerconstant, Integerconstant) 'OF' Variable
95)	Proceduredec	=	Answerspecoption Recursivity Prochead ; Statement
91)	Procedurestatement	=	Variable
99)	Prochead	=	Identifier
			Identifier (Parameterspeclist)
104)	Proctype	=	Answerspecoption Recursivity
			Answerspecoption Recursivity (Parameterspeclist)

2)	Range	=	Bound 'TO' Bound
			Bound : Bound
79)	Rangetype	=	'INTEGER' (Range)
			'STATUS' (Identifierlist)
			Usertype
40)	Readonlyoption	=	'READONLY' Identifierlist
			'READONLY' 'OPEN'
			Empty
22)	Real	=	Digitlist . Digitlist$_{10}$Signedinteger
			Digitlist . Digitlist
			Digitlist$_{10}$Signedinteger
			$_{10}$Signedinteger
98)	Recursivity	=	'RECURSIVE'
			'PROCEDURE'
9)	Referencetype	=	'REF' Directtype
			'REF' 'CONST' Directtype
23)	Signedinteger	=	Addoperator Integer
			Integer
55)	Simpleexpression	=	Term
			Addoperator Term
			Simpleexpression Addoperator Term
1)	Simpletype	=	'INTEGER'
			'INTEGER' (Range)
			'FIXED' (Size , Fractionbits)
			'FLOATING'
			'BYTE'
			'STATUS' (Identifierlist)
			Usertype
4)	Size	=	Integerconstant

67)	Statement	=	Block
			Assignmentstatement
			Conditionalstatement
			Loopstatement
			Casestatement
			Procedurestatement
			Dummystatement
			Label : Statement
			'GOTO' Label
			'ANSWER' Expression
			'RETURN'
			'REPEAT'
			'ESCAPE'
			Mascotstatement
71)	Statementsequence	=	Statement
			Statement ; Statementsequence
27)	String	=	"Stringitemlist"
29)	Stringitem	=	Character other than quotation marks
			" "
28)	Stringitemlist	=	Stringitemlist Stringitem
			Empty
63)	Subcondition	=	Conditionelement
			Subcondition 'AND' Conditionelement
84)	Tail	=	'THEN' Statementsequence 'ENDLOOP'
			'ENDLOOP'
56)	Term	=	Factor
			Term Multoperator Factor
83)	Tooption	=	'TO' Expression
			Empty
30)	Typedec	=	'TYPE' Usertype = Typedefiner
31)	Typedefiner	=	Declarabletype
			Proctype
			Groupdefinition

20)	Unsignedconstant	=	Number
			Identifier
6)	Usertype	=	'Identifier'
52)	Value	=	Number
			String
			'NIL'
			Identifier
			Display
50)	Variable	=	Identifier
			Variable . Identifier
			Variable (Parameterpack)
			[Variable]
87)	Whenelement	=	'WHEN' Cases : Statementsequence
86)	Whenset	=	Lastwhen
			Whenelement
			Whenelement Whenset

SOFTWARE SCIENCES LIMITED
Abbey House
282/292 Farnborough Road
Farnborough Hampshire
Telephone: 0252 44321
Telex: 858228
Cables: Softwares Farnborough Hampshire

London & Manchester House
Park Street
Macclesfield
Cheshire SK11 6SR
Telephone Macclesfield 29241-4

Software Sciences Nederland B.V.
Reguliersdwarsstraat 9
Amsterdam
NETHERLANDS
Telephone Amsterdam 235631
Telex 17029