THE MITRE CORPORATION
BEDFORD, MASS.

MEMORANDUM

TO: E. L. Burke DATE: 3 February 1975

FROM: K. J. Biba, W. L. Schiller MEMO NO.: D73-M750

SUBJECT: Multics Design Note #8 - A Specification Language

COPIES TO: Distribution

Context

The proliferation of "styles" within the Multics security kernel specification has lead to an attempt to standardize (and formalize) a syntax for specifications. Further, the anticipated size of the hierarchical kernel specification requires the existence of automated tools to cope with its management and analysis. Such tools require a precise language definition. Current work is addressing the formal definition of such a language, as well as the identification and construction of necessary tools. This note addresses a rather informal (and preliminary) definition of the proposed specification style (and language). Examples corresponding closely to the style advocated in this note may be found in the specifications of Multics Design Note #7.

Style Description

A specification may be considered as a series of clauses; each clause defining one portion of the specification. Each clause is composed of one or more statements; each statement defining a boolean relation appropriate for the clause. Each clause begins with a clause label (identifying the type of clause) followed by a statement list.

```
<specification> ::= <clause_list>

<clause_list>  ::= <clause> |
                   <clause>  <clause_list>

<clause>  ::= <clause_label>  ':'  <statement_list> ';'

<statement_list> ::= <statement> |
                     <statement_list> ';'  <statement>
```

Eight clause types have been identified:

1) function identification,
2) statement of purpose,
3) initial value definition,
4) possible value (range) definition,
5) explicit value definition,
6) statement of effect,
7) abbreviation declaration, and
8) statement of exceptions.

While no clause ordering is apparent above, we require the head of a specification to be a "function" clause. Function clauses serve to delimit one function from another. No nesting of function specifications is thus permitted.

The function clause expects only one component statement: a <function_name> statement.

```
<function_name> ::=  <name> |
                     <name>  '('  <parameter_list> ')'

<parameter_list> ::=  <name>|
                      <name> ','  <parameter_list>
```

The <clause_label> may have one of four values (at this time) denoting the type of function being specified:

1) "V_function" denoting a value returning function;
2) "Hidden_V_function" denoting a value returning function which may not be invoked by the user of the specified module;
3) "O_functions" denoting operations which effect changes of state (defined by V_functions); and
4) "OV_function" denoting a function which both effects a state change and returns a value.

The purpose clause provides a narrative description of the function's purpose. Its <clause_label> is "Purpose" and the <statement> is any character string not containing ";".

The initial value clause defines the initial value of all types of V_functions. Its <clause_label> is "Possibly". The single statement of this clause is an uninterpreted character string which defines the function's range.

The value clause defines the value of a V_function. It should be constituted by a single statement composed of an expression of function names and parameters yielding a value in the range specified in the possible value clause.

The abbreviation clause (whose <clause_label> is "Let") acts as a set of text macro definitions. Its intent is to allow the substitution of short, locally (within a single function) defined names for longer expressions of functions, liberals and parameters. This clause is composed of one or more statements each of the form:

```
<statement>  ::=  <name>  '='  <expression>.
```

The exception clause defines conditions under which the associated function will abort (before performing its intended function, either value_returning or state change). The clause_label is "Exception_if". The clause is composed of one or more statements of the form:

<statement> ::= <text_label> '!' <expression>.

The token <text_label> represents a symbolic name for the particular exception which may be interpreted as an error case. <Expression> is a boolean valued expression composed of function, parameters, literals and operators. All exception conditions must evaluate to boolean "false" if the associated function is to be invoked.

The effect clause defines conditions that must hold at the termination of an O_function. The <clause_label> is "Effect". It is composed of one or more relational statements which express conditions that must hold upon the value of V_functions at the termination of the O_function. Variants of "if" and "case" statements are specifically proposed to express implications.

Literals (including "true" and "false") are expressed as:

<literal> ::= '"' <texting_string> '"'.

Integers do not require enclosing quotes.

Function and parameter names are defined by:

```
<name> ::=  <letter> |
            <name> <letter> |
            <name> <number> |
            <name> '_' |
            <name> '.' |
            <name> '#' |
```

No specific rules for capitalization are proposed. Both upper and lower case are acceptable. Likewise, no specific rules for indentation will be formulated, though its judicious use is highly encouraged.

The following words (in any combinations of case) are reserved.

O_function	let
V_function	exception_if
OV_function	if
hidden	then
purpose	else
initially	endif (fi/end)

possibly

value

effect

caseof

endcase (esac/end

K. J. Biba
Intelligence and
Information Systems

W. L. Schiller
Intelligence and
Information Systems

KJB/WLS:jkl

Distribution:
S. R. Ames, Jr.
D. E. Bell
E. H. Bensley
K. J. Biba
E. L. Burke
C. S. Chandersekaran
M. Gasser
C. D. Jordan
L. J. LaPadula
S. B. Lipner
J. K. Millen
R. D. Rhode
W. L. Schiller
D. F. Stork
J. C. C. White