

Specification for

**Computer
programming language
RTL/2**

UDC 681.3.06RTL/2

NO COPYING WITHOUT BSI PERMISSION EXCEPT AS PERMITTED BY COPYRIGHT LAW



Document Sharing Hub

Cooperating organizations

The Data Processing Standards Committee, under whose direction this British Standard was prepared, consists of representatives from the following Government departments and scientific and industrial organizations:

British Computer Society Ltd.*
 British Equipment Trade Association*
 British Paper and Board Industry Federation (PIF)
 British Printing Industries Federation
 Central Computer Agency (Civil Service Department)*
 Committee of London Clearing Bankers on behalf of the Committee of Scottish Clearing Bankers,
 Cooperative Bank, Central Trustee Savings Bank and Yorkshire Bank
 Department of Industry (Computers Systems and Electronics)
 Department of Industry (National Physical Laboratory)*
 Electricity Supply Industry in England and Wales*
 Government Communications Headquarters
 HM Customs and Excise
 Institute of Cost and Management Accountants
 Institute of Purchasing and Supply
 Institution of Electrical Engineers
 Institution of Mechanical Engineers
 Inter-university Committee on Computing
 London Transport Executive
 Ministry of Defence*
 National Computer Users' Forum
 National Computing Centre Ltd.*
 National Research Development Corporation
 Post Office*
 Society of British Aerospace Companies Limited

The organizations with an asterisk in the above list, together with the following, were directly represented on the committee entrusted with the preparation of this British Standard:

Association for Literary and Linguistic Computing
 Association of Computer Units in Colleges of Higher Education (ACUCHE)
 British Gas Corporation
 Computing Services Association
 Control and Automation Manufacturers' Association (BEAMA)
 Edinburgh Regional Computing Centre
 Engineering Equipment Users' Association
 Hatfield Polytechnic
 University of London

This British Standard, having been prepared under the direction of the Data Processing Standards Committee, was published under the authority of the Executive Board and comes into effect on 30 September 1980

© BSI 03-2000

The following BSI references relate to the work on this standard:
 Committee reference DPS/13
 Draft for comment 78/64581 DC

ISBN 0 580 11441 4

Amendments issued since publication

Amd. No.	Date of issue	Comments
3722	July 1981	
8233	October 1994	Indicated by a sideline in the margin

Contents

	Page
Cooperating organizations	Inside front cover
Foreword	ii
1 Scope	1
2 Reference	1
3 Definitions	1
4 Syntax	1
4.1 Syntactic metalanguage	1
4.2 Example	1
4.3 Syntax requirements	1
5 Examples	5
6 Compliance	5
6.1 Implementations	5
6.2 Programs	6
7 Requirements	6
7.1 Basic elements	6
7.2 Declarations	12
7.3 Expressions	22
7.4 Statements	33
7.5 Modules	40
7.6 Integrity	41
Appendix A Information on input/output	43
Appendix B Information on error recovery	44
Appendix C Recommendation on compiler limits	44
Figure 1 — Numerical mode conversion in conditional expressions	25
Figure 2 — Numerical mode conversion for operands	27
Table 1 — Alphabetical lists of syntax requirements	2
Table 1A — Item syntax	2
Table 1B — Module syntax	3
Table 2 — RTL/2 character set	7
Table 3 — Keywords	9
Table 4 — Monadic operators, operands and results	28
Table 5 — Dyadic operators, operands, results and precedence	29
Table 6 — Form of result of use of shift operators	30
Publications referred to	Inside back cover

Foreword

This standard has been prepared under the direction of the Data Processing Standards Committee and extracts in this standard from “RTL/2 Language Specification”, Version 2, ICI, 1974 are reproduced by permission of Imperial Chemical Industries Limited.

In drafting this standard the continued stability of RTL/2 has been a prime objective. However, apart from changes to clarify or correct the specification, a few minor alterations have seemed advisable. These affect the scope of record component names, strings, LET definitions, equality of label values and the character set. These alterations have been carefully designed to correct features of the original specification that with hindsight have proved to be inappropriate and yet do not alter the semantics of valid programs.

RTL/2 provides a method for writing both application and system programs for use in real-time computing and is especially suited for on-line data acquisition, communication and control systems.

The overall objectives of the language are:

- a) to reduce the direct cost of the development and maintenance of software;
- b) to encourage the creation of more reliable systems;
- c) to increase the mobility of programming staff;
- d) to provide continuity of method and flexibility in choice of equipment by ensuring the portability of application programs.

RTL/2 is designed to be as machine-independent as practicable even at the expense of certain (probably obsolescent) machine architectures. RTL/2 recognizes, however, that operating systems need a degree of flexibility that is incompatible with the security needs of application programs. Accordingly, RTL/2 comprises two languages, the full language being the system language, and a secure subset (see 7.6) being the application language.

It is virtually impossible to define precisely a high-level language so that programs will be executed equally efficiently by all types of computer. It has therefore been necessary to leave certain areas of RTL/2 deliberately unspecified so that implementations can take the best advantage of characteristics of particular computers. The most important of these areas are:

- a) accuracy and range of real values;
- b) number of bits in an integer word;
- c) behaviour on arithmetic overflow.

Such areas are indicated in this standard by the use of phrases such as “implementation-dependent”. On the other hand, it should be noted that the representation of integer values has been explicitly specified to be in twos complement form; thus implementations of RTL/2 on computers that do not use this representation will be less efficient.

Program structure. A computer that is not specifically designed to execute RTL/2 code directly would need to be enhanced with various control routines to create an RTL/2 machine. In a simple single-state machine, the rest of the software, if derived from RTL/2, could be of items of software of similar status and this group of items would be known as a “program complex”. More usually, however, and of necessity in a two-state machine, the remaining software has a hierarchical structure. The top level, usually known as the “supervisor”, is itself a program complex and the individual groups of lower levels can themselves be considered to be program complexes, but with an environment different from that of the supervisor complex.

An RTL/2 complex consists of a collection of items known as bricks, of which there are the following types:

- a) procedure brick;
- b) data brick;
- c) stack brick.

A procedure brick consists of the declaration of a single literal procedure. A procedure is a read-only piece of code describing an executable process, and may have parameters and local variables, but the latter are restricted to be scalars. The entry mechanism and implementation of local variables is reentrant. The coding of a procedure may directly access variables in a data brick, but not the local variables or parameters of another procedure. A procedure may not include internal procedures.

A data brick is a named static collection of scalars, arrays and records.

A stack brick consists of the declaration of a single literal stack. A stack is an area used for the storage of links, dynamic (i.e. local) variables and other housekeeping items.

Several bricks may be grouped together to form a module that is the unit of compilation. A program complex is the result of linking together one or more such modules.

The compiler needs to be informed of the environment of a module for satisfactory compilation to take place. This environment, in general, consists of two parts; first there is the environment of the complex as a whole and secondly there are interfaces with other modules in the complex.

The environment of the complex as a whole is, in the case of the supervisor, simply the enhanced machine, whereas in the case of a complex running under the supervisor in a two-state machine, it also consists of the environment provided by that supervisor. This environment comprises a set of procedures accessed as supervisor calls (SVC procedures) plus a set of data bricks private to each task and nominally housekept by the supervisor (SVC data). A module needs to include descriptions of SVC procedures and data bricks that it accesses.

To reference a brick within another module in the complex, a description of the brick needs to be included in the referring module, and the brick referred to in the other module needs to have been specified as an external entry in that module.

The various cross-references implied by these environment descriptions are satisfied by the linker program (see 7.5).

Multitasking. A program complex may represent a simple program with a definite start and finish. It could however consist of the code and data for several processes running concurrently, and in order to describe such a complex the term “task” is introduced.

A task, broadly speaking, is an identifiable execution of a logically coherent set of instructions by a (pseudo-) processor.

A conventional computer with a single processor can be considered to be obeying at all times one unique task. It is, however, usually more convenient to consider the execution of each logically distinct process as a task and to convert the one actual processor into several psuedo-processors by a scheduling algorithm within a supervisor.

As a language, RTL/2 imposes few constraints on the design of multitasking systems and the actual facilities of such systems are outside the scope of this standard. It is, however, possible to describe in outline the intended relationships between the various bricks in a multitasking system.

The creation, control and elimination of tasks will, in general, be performed by a supervisor, and the supervisor call (see 7.5) will provide the channel for the communication of task and control requests.

Whenever a new task is created, a stack will be nominated as workspace and a procedure will be nominated as the coding to be obeyed. Later operations on the task may be made by reference to its stack. Each stack can, of course, be used only by one task at a time, whereas procedures and data bricks may be used by several concurrent tasks. The procedure nominated as coding can call other procedures and access variables in data bricks as required.

Communication between tasks can be via supervisor calls and any message scheme provided by the supervisor, or simply via data bricks with the suitable use of semaphores.

An SVC data brick is private to each task and may be used in a reentrant manner. It will be housekept on task changes and this might well be implemented by mapping it on to part of the stack.

Editorial note. It is normal convention in British Standards to use italic type for algebraic quantities. Since the status of such quantities contained in this standard may or may not directly represent true variable quantities, this convention has not been adopted in this standard.

A British Standard does not purport to include all the necessary provisions of a contract. Users of British Standards are responsible for their correct application.

Compliance with a British Standard does not of itself confer immunity from legal obligations.

Summary of pages

This document comprises a front cover, an inside front cover, pages i to iv, pages 1 to 44, an inside back cover and a back cover.

This standard has been updated (see copyright date) and may have had amendments incorporated. This will be indicated in the amendment table on the inside front cover.

1 Scope

This British Standard specifies the semantics and syntax of the computer programming language RTL/2 by specifying requirements for a compiler and for a conforming program.

NOTE The specification requirements have been drafted so that all features of RTL/2 are either explicitly defined, or explicitly stated to have been left intentionally undefined.

Appendix A contains information regarding input/output. Appendix B contains information on error recovery. Appendix C makes a recommendation on compiler limits.

2 Reference

The title of the publication referred to in this standard is given on the inside back cover.

3 Definitions

For the purposes of this British Standard the definitions given in BS 3527 apply.

4 Syntax

4.1 Syntactic metalanguage¹⁾. The notation used in this standard to specify the syntax of the language is as follows.

- a) The terminal symbols of the language denote themselves. The names of classes are in lower case letters whereas the language alphabet is represented by the upper case letters. Items are separated from each other by spaces where necessary.
- b) The metasymbol `:=` denotes “is”, and the metasymbol `|` denotes “or”.
- c) The brackets `[` and `]` are used to denote that the items enclosed within are optional.
- d) A sequence of three dots `...` denotes that the immediately preceding item may be repeated many times.

4.2 Example. The following example denotes that a “name” is a sequence of letters and digits of which the first is a letter. A “digit” is one of 0, 1 9 and a “letter” is one of A, B Z. The individual digits and letters are terminal symbols and cannot be decomposed further.

name `:= letter [ letter | digit ] ...`

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

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`

In the text, the names of classes are enclosed in quotes when formal correctness is emphasized. In less formal passages the quotes are omitted.

4.3 Syntax requirements. The syntax requirements, with references to relevant text in clause 7, are specified in Table 1.

NOTE The syntax requirements listed in Table 1 should be interpreted in a synthetic rather than analytic way. It is intended primarily for the user of this standard to determine the form in which a program is to be written and reference to this standard by a compiler, to determine whether a form is valid, is a secondary consideration. In some cases involving the class “identifier”, the syntax is ambiguous (i.e. it allows two parses to some forms), and semantic information (the mode of the identifier) is required to determine which parse should be used. These cases are as follows.

- a) In the production of “arrayelement”, both “variable” and “array” can be resolved to “identifier”.
- b) In the production of “recordcomponent”, both “variable” and “record” can be resolved to “identifier”.
- c) In the production of “constant”, both “identifier” and “variable” can be resolved to “identifier”.
- d) In the production of “primary”, both “functioncall” and “variable” can be resolved to “identifier ([expn [, expn] ...])”.

The syntax of RTL/2 is presented at two levels. At the higher level the root is “module” and the terminal symbols are the items; at this level the layout characters may be freely inserted to increase legibility. At the lower level the root is “item” and the terminal symbols are characters; at this level the layout characters are generally significant and cannot be arbitrarily inserted. The class “item” is not used other than as a descriptive convenience.

¹⁾ A British Standard syntactic metalanguage is at an early stage of preparation.

Table 1 — Alphabetical lists of syntax requirements

Table 1A — Item syntax

Requirement		Reference
bindigit	::= 0 1	7.1.3
bindigitlist	::= bindigit ...	7.1.3
codeheading	::= CODE digitlist, digitlist;	7.1.8
codeitem	::= character-other-than-trip1-or-trip2 trip1 letitem trip2 name	7.1.8
codeseq	::= codeheading codeitem ...	7.1.8
comment	::= % sequence-of-RTL/2-characters-excluding-%-and-newline %	7.1.5
compoundseparator	::= := :/ / <= >= : : :# :	7.1.9
digit	::= 0 1 2 3 4 5 6 7 8 9	7.1.2
digitlist	::= digit ...	7.1.3
exponent	::= E sign digitlist	7.1.3
fraction	::= real B sign digitlist	7.1.3
hexdigit	::= digit A B C D E F	7.1.3
hexdigitlist	::= hexdigit ...	7.1.3
integer	::= digitlist BIN bindigitlist OCT octdigitlist HEX hexdigitlist "stringchar"	7.1.3
item	::= letitem title option codeseq	7.1.1
letdefinition	::= LET name = [letitem-other-than-semicolon] ...	7.1.11
letitem	::= name number string comment separator	7.1.11
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	7.1.2
name	::= letter [letter digit \$ _] ...	7.1.2
number	::= real integer fraction	7.1.3
octdigit	::= 0 1 2 3 4 5 6 7	7.1.3
octdigitlist	::= octdigit ...	7.1.3
opchar	::= letter digit	7.1.7
opitem	::= opchar...	7.1.7
real	::= digitlist, digitlist [exponent] digitlist exponent	7.1.3
separator	::= compoundseparator simpleseparator	7.1.9
sign	::= [+ -]	7.1.3
simpleseparator	::= () * + - , . / : ; < = > #	7.1.9
string	::= " [stringchar ... stringinsert] ..."	7.1.4
stringchar	::= RTL/2-character-other-than-"- # HT-LF	7.1.3
stringinsert	::= # [stringitem [, stringitem] ...] #	7.1.4
stringitem	::= integer [(length)]	7.1.4
title	::= TITLE sequence-of-RTL/2-characters-excluding-semicolon	7.1.6
trip1	::= first-code-trip-character	7.1.8
trip2	::= second-code-trip-character	7.1.8

Table 1 — Alphabetical lists of syntax requirements

Table 1B — Module syntax

Requirement		Reference
amode	::= simplemode recmode	7.2.4.3
arraydec	::= ARRAY (length [, length] ...) amode initidlist	7.2.4.3
array	::= identifier recordcomponent	7.3.2.3
arrayelement	::= variable (subscriptlist) array (subscriptlist)	7.3.2.3
arrayinititem	::= datainitvalue [(length)]	7.2.6.5
arrayinitvalue	::= ([arrayinititem [, arrayinititem] ...]) string	7.2.6.5
arraymode	::= ARRAY [([,]...)] amode	7.2.4.3
arraysimplespec	::= ARRAY (length [, length] ...) simplemode idlist	7.2.5.1
arrayspec	::= ARRAY (length [, length] ...) amode idlist	7.5.1
assignmentst	::= destination := [destination :=] ... expn	7.4.4
block	::= BLOCK blockbody ENDBLOCK	7.4.3
blockbody	::= [simpledec;] ... sequence	7.2.7.1
brick	::= [ENT] datadec [ENT] procdec [ENT] stackdec	7.5.1
codest	::= codeseq	7.4.13
comparator	::= = # < <= > >= := :#	7.3.4
comparison	::= expn comparator expn	7.3.4
condexpn	::= IF condition THEN expn [ELSEIF condition THEN expn] ... ELSE expn END	7.3.2.5
condition	::= subcondition [OR subcondition] ...	7.3.4
constant	::= number identifier variable structure string	7.3.2.2
datadec	::= DATA identifier; declaration [; declaration] ... ENDDATA	7.2.11
datadescn	::= DATA identifier; dspec [; dspec] ... ENDDATA	7.5.1
datainitvalue	::= primitivvalue refinitvalue recinitvalue arrayinitvalue	7.2.6.5
declaration	::= [simpledec arraydec recorddec]	7.2.11
destination	::= variable VAL variable	7.4.4
dspec	::= [simplespec arrayspec recordspec]	7.5.1
dyadicop	::= SLL SRL SHL SLA SRA SHA * / // :/ MOD LAND LOR NEV + -	7.3.3.4
dummyst	::= []	7.4.12
enviromdescn	::= EXT stackdescn EXT procdescn EXT datadescn SVC procdescn SVC datadescn	7.5.1
expn	::= term [dyadicop term] ...	7.3.3.4
forst	::= [FOR identifier: = expn [BY expn]] TO expn DO blockbody REP	7.4.8
functioncall	::= variable paralist identifier paralist	7.3.2.4
gotost	::= GOTO expn	7.4.5
identifier	::= name	7.1.2
idlist	::= identifier [, identifier] ...	7.2.5.1
ifst	::= IF condition THEN sequence [ELSEIF condition THEN sequence] ... [ELSE sequence] END	7.4.7
initidlist	::= inititem [, inititem] ...	7.2.3
inititem	::= identifier [:= [identifier: =] ... initvalue]	7.2.3
initvalue	::= datainitvalue localinitvalue	7.2.6.5
labellist	::= identifier [, identifier] ...	7.4.6
labels	::= [identifier;] ...	7.4.2
length	::= staticintexpn	7.2.4.2
localinitvalue	::= expn	7.2.6.5
module	::= moduleitem [; moduleitem] ...	7.5.1
moduleitem	::= [enviromdescn recmodedef letdefinition title brick]	7.5.1
monadicop	::= + - ABS NOT REAL INT FRAC BYTE LENGTH	7.3.3.3

Table 1 — Alphabetical lists of syntax requirements

Table 1B — Module syntax

	Requirement	Reference
paradescription	::= [pspec [, pspec] ...]	7.2.7.1
paralist	::= ([expn [, expn] ...])	7.3.2.4
plainmode	::= REAL INT FRAC BYTE	7.2.3
primary	::= constant variable functioncall condexpn (expn)	7.3.3.1
primitivalue	::= sign number identifier	7.2.6.2
primmode	::= plainmode progmode	7.2.3
procdec	::= PROC identifier (paradescription) resultmode; blockbody ENDPROC	7.2.7.1
procdescn	::= PROC procdescriptor idlist	7.5.1
procdescriptor	::= ([simplemode [, simplemode] ...]) resultmode	7.2.7.3
procst	::= variable paralist identifier paralist	7.4.10
progmode	::= LABEL STACK PROC procdescriptor	7.2.3
pspec	::= simplespec	7.2.7.1
recinitvalue	::= (datainitvalue [, datainitvalue] ...)	7.2.6.5
recmode	::= recmodeident	7.2.5.1
recmodedef	::= MODE recmodeident (rspec [, rspec] ...)	7.2.5.1
recmodeident	::= identifier	7.2.5.1
record	::= identifier arrayelement	7.3.2.3
recordcomponent	::= variable, selector record.selector	7.3.2.3
recorddec	::= recmode initidlist	7.2.5.2
recordspec	::= recmode idlist	7.5.1
refinitvalue	::= variable structure string	7.2.6.3
resultmode	::= [simplemode]	7.2.7.2
returnst	::= RETURN RETURN (expn)	7.4.11
rspec	::= simplespec arraysimplespec	7.2.5.1
selector	::= identifier	7.3.2.3
sequence	::= statement [; statement] ...	7.4.1
simpledec	::= simplemode initidlist	7.2.3
simplemode	::= [REF] primmode REF arraymode REF recmode	7.2.3
simplespec	::= simplemode idlist	7.2.5.1
stackdec	::= STACK identifier length	7.2.8
stackdescn	::= STACK idlist	7.5.1
statement	::= labels unlabelledst	7.4.1
staticdyadicop	::= + - * /	7.3.7
staticintexpn	::= staticterm [staticdyadicop staticterm]	7.3.7
staticmonadicop	::= + -	7.3.7
staticprimary	::= integer (staticintexpn)	7.3.7
staticterm	::= [staticmonadicop] ... staticprimary	7.3.7
structure	::= array record	7.3.2.3
subcondition	::= comparison [AND comparison] ...	7.3.4
subscriptlist	::= expn [, expn] ...	7.3.2.3
switchst	::= SWITCH expn OF labellist	7.4.6
term	::= [monadicop] ... primary	7.3.3.3
unlabelledst	::= block assignmentst gotost switchst ifst forst whilest procst returnst dummys codest	7.4.1
variable	::= identifier arrayelement recordcomponent	7.3.2.3
whilest	::= WHILE condition DO sequence REP	7.4.9

NOTE The class names “npconstant”, “npexpn” and “vectordec” have been omitted from Table 1 because they are merely subclasses of “constant”, “expn” and “arraydec” respectively; these subclasses are introduced in this standard only as an explanatory aid.

5 Examples

This standard contains many examples in which, for compactness, identifiers are used without having been formally declared. In all such cases the identifiers should be considered to be declared as follows.

```

MODE COMPLEX (REAL RL, IM);
MODE LIST (INT HD, REF LIST TL);
MODE PERSON (INT AGE, ARRAY (8) BYTE NAME, REF ARRAY BYTE ADDRESS, REF PERSON
MOTHER, FATHER, REF ARRAY PERSON CHILDREN, SIBLINGS, BYTE SEX);
EXT PROC () REAL TIME;
EXT PROC (REAL) REAL LOG, EXP, SIN, COS;
EXT PROC (INT, INT, INT) INT F;
EXT PROC (REF ARRAY BYTE) TWRT;
EXT PROC (INT) IWRT;
DATA EXAMPLES;
  REAL X, Y, Z;
  REF REAL XX, YY, ZZ;
  INT I,J, K, L;
  REF INT JJ, KK, LL;
  BYTE M, N;
  FRAC P, Q, R;
  LABEL RESTART;
  COMPLEX U, V, W;
  LIST CELLA, CELLB;
  REF LIST NEXTCELL;
  PERSON JOHN, JIM, JANE;
  REF PERSON WHO;
  ARRAY (7)INT G;
  ARRAY (10) REAL A, B, C;
  ARRAY (5) LIST CELLS;
  ARRAY (100) PERSON PEOPLE;
  ARRAY (5, 10) REAL A2, B2, C2;
  ARRAY (6) LABEL S;
  REF ARRAY REAL AA, BB;
  REF ARRAY (,) REAL AA2, BB2;
  PROC () ROUTINE;
  PROC (REAL) REAL FN;
ENDDATA

```

6 Compliance

6.1 Implementations

6.1.1 An implementation of RTL/2 that can accept all of the features of the language specified in clause 7 and with the meanings defined therein complies with the requirements of this standard.

6.1.2 An implementation that omits any of the features of the language specified in clause 7, or that includes such a feature but with a meaning altered in any way from that specified in clause 7, does not comply with the requirements of this standard.

6.1.3 An implementation that includes features additional to those specified in clause 7 (i.e. extensions) complies with the requirements of this standard provided that:

- a) the extensions do not singly or collectively alter in any way the meanings of the features specified in clause 7; *and*
- b) the extensions are clearly detailed in all descriptions of the implementation as being “extensions to RTL/2 as specified by BS 5904”; *and*
- c) the compiler is capable of issuing a warning message at each use of such an extension.

6.2 Programs

6.2.1 A program written in RTL/2 complies with the requirements of this standard provided that the program uses only those features of the language specified in clause 7, together with any desired sequences of machine code included in the manner specified in clause 7.

6.2.2 Any program that includes any extension to the language (see 6.1.3), or includes machine code in a manner other than as allowed by the requirements of clause 7 does not comply with the requirements of this standard.

7 Requirements

7.1 Basic elements

7.1.1 Items. The text of an RTL/2 program shall be a sequence of characters drawn from the character set specified in Table 2. No other character shall appear in the text of an RTL/2 program, except within a code sequence (see 7.1.8). The three characters #, £ and \$ shall be interchangeable and all shall mean the same thing (except possibly within a code sequence).

NOTE 1 For simplicity, only the character # is shown in the syntax requirements in this standard.

NOTE 2 The characters are grouped together into items of various sorts and the program text is best considered as a sequence of these items. 7.1 describes this grouping which should be considered to occur before any further analysis takes place.

The characters shall be grouped into items. Each item shall be one of the following:

- a) name;
- b) arithmetic constant;
- c) string;
- d) comment;
- e) title;
- f) code sequence;
- g) separator.

An item shall be terminated by any character that cannot be interpreted as being part of that item.

NOTE 3 As a consequence of this general requirement, the layout characters space, newline or tab terminate most items. Layout characters are otherwise not significant outside items and may be freely used to improve the legibility of program text.

Items can be divided into two categories according to whether the requirements of 7.1.11 allow them to appear in the replacement sequence of a LET definition.

The syntax shall be:

item ::= letitem | title | option | codeseq

Table 2 — RTL/2 character set

Character	Decimal value	Language use	Reference
HT	9	layout: horizontal tab	7.1.1
LF	10	layout: newline	7.1.1
SP	32	layout: space	7.1.1
!	33	not used	
"	34	string quote	7.1.4
#	35	not equals, strings	7.1.4 and 7.3.4
\$	36	names	7.1.2
%	37	comments	7.1.5
&	38	not used	
'	39	byte quote	7.1.3
(	40	open bracket	
)	41	close bracket	
*	42	multiply	7.3.3.4
+	43	add	7.3.3.4
,	44	comma	
—	45	minus	7.3.3.4
.	46	constants, records	7.1.3 and 7.3.2.3
/	47	divide	7.3.3.4
0 to 9	48 to 57	numbers	7.1.3
:	58	labels, assignment, etc.	7.4.2, 7.4.4 and 7.3.4
;	59	statement, declaration separator	
<	60	less than	7.3.4
=	61	assignment, equals	7.4.4 and 7.3.4
>	62	greater than	7.3.4
?	63	not used	
@	64	not used	
A to Z	65 to 90	names, numbers	7.1.2 and 7.1.3
[	91	not used	
\	92	not used	
]	93	not used	
↑	94	not used	
—	95	names	
'	96	not used	
a to z	97 to 122	names	
{	123	not used	
	124	not used	
}	125	not used	
~	126	not used	
	160 to 255	not used	

NOTE The characters designated as not used have no particular purpose in the language but they may occur in strings, comments and titles.

7.1.2 Names. A name shall consist of a sequence of letters, digits, underscore and dollar characters of which the first is a letter. There shall be no limit to the number of characters in a name and all the characters shall be significant.

Names of EXT, ENT and SVC bricks (see 7.5) shall not be specially restricted and shall be fully significant within a module.

NOTE 1 However, the linkage system may impose restrictions on the number of characters that are externally significant and hence on the user's choice of such names.

The syntax shall be:

```

name      ::= letter [ 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
digit     ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
identifier ::= name

```

Lower-case letters may be used in names. Names that differ only in their use of upper-case and lower-case letters are considered identical.

Names shall be used for two purposes.

a) To denote the language keywords, i.e. names such as IF and REAL, that have a predefined meaning. All the keywords are listed in Table 3.

b) To denote user identifiers, used for a variety of purposes such as naming variables, arrays of variables or pieces of code. A name that denotes a keyword shall not be used as an identifier.

Examples of “identifier” are:

```

J
GEORGE
V17X
NO2

```

NOTE 2 As a consequence of the general rules regarding the termination of items, a layout character will terminate a name. Thus PROC FRED consists of two adjacent names whereas PROCFRED is a single name. Similarly, it is essential that the keyword GOTO should not be written as GO TO.

7.1.3 Arithmetic constants. The four types of numerical data in RTL/2 shall be “real”, “integer”, “fraction” and “byte”; these are specified in detail in 7.2.2.

Arithmetic constants (numbers) shall be used to denote literal and initial values of these types.

NOTE 1 The form “integer” may be used to denote constants of both type integer and byte (see 7.3.2.2).

The syntax shall be:

```

number    ::= real | integer | fraction
sign      ::= [ + | - ]
exponent  ::= E sign digitlist
real      ::= digitlist . digitlist [ exponent ] | digitlist exponent
integer   ::= digitlist | BIN bindigitlist | OCT octdigitlist | HEX
           hexdigitlist | “stringchar”
fraction  ::= real B sign digitlist
digitlist ::= digit ...
bindigitlist ::= bindigit ...
octdigitlist ::= octdigit ...
hexdigitlist ::= hexdigit ...
bindigit  ::= 0 | 1
octdigit  ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
hexdigit  ::= digit | A | B | C | D | E | F
stringchar ::= RTL/2-character-other-than-” -# -HT-LF

```

Lower-case letters e, b and a – f may be used in the forms for exponent, fraction and hexdigit respectively.

NOTE 2 A stringchar may be an actual space.

The numerical value of a constant denoted by a stringchar in single quotes shall be the decimal value of that character specified in Table 2.

Table 3 — Keywords

Keyword	Reference	Keyword	Reference	Keyword	Reference
ABS	7.3.3.3	FRAC	7.2.3 and 7.3.3.3	REAL	7.2.3 and 7.3.3.3
AND	7.3.4	GOTO	7.4.5	REF	7.2.3
ARRAY	7.2.4	HEX	7.1.3	REP	7.4.8 and 7.4.9
BIN	7.1.3	IF	7.3.2.5 and 7.4.7	RETURN	7.4.11
BLOCK	7.4.3	INT	7.2.3 and 7.3.3.3	RTL	7.1.8
BY	7.4.8	LABEL	7.2.3	SHA	7.3.3.4
BYTE	7.2.3 and 7.3.3.3	LAND	7.3.3.4	SHL	7.3.3.4
CODE	7.1.8	LENGTH	7.3.3.3	SLA	7.3.3.4
DATA	7.2.11	LET	7.1.11	SLL	7.3.3.4
DO	7.4.8 and 7.4.9	LOR	7.3.3.4	SRA	7.3.3.4
ELSE	7.3.2.5 and 7.4.7	MOD	7.3.3.4	SRL	7.3.3.4
ELSEIF	7.3.2.5 and 7.4.7	MODE	7.2.5.1	STACK	7.2.3, 7.2.8 and 7.5.1
END	7.3.2.5 and 7.4.7	NEV	7.3.3.4	SVC	7.5.1
ENDBLOCK	7.4.3	NOT	7.3.3.3	SWITCH	7.4.6
ENDDATA	7.2.11	OCT	7.1.3	THEN	7.3.2.5 and 7.4.7
ENDPROC	7.2.7.1	OF	7.4.6	TITLE	7.1.6
ENT	7.5.1	OPTION	7.1.7	TO	7.4.8
EXT	7.5.1	OR	7.3.4	VAL	7.4.4
FOR	7.4.8	PROC	7.2.3, 7.2.7.1 and 7.5.1	WHILE	7.4.9

NOTE The following additional keywords are reserved for use in extensions (see 6.1.3): CASE ENUM EXITDO MODE_CONVERT PRAGMA WHEN

In the case of the binary, octal and hexadecimal forms for integers, the keyword BIN, OCT or HEX shall be followed by one or more of the layout characters space, newline or tab. All other occurrences of a layout character shall terminate an arithmetic constant.

The fraction constant shall include a binary scale factor following B, e.g. $17.6B - 5$ is stored as 17.6×2^{-5} .

NOTE 3 A real constant may include a decimal exponent following E, e.g. $0.1 E7$ is stored as 0.1×10^7 .

The following are examples of “number”:

3.47	0.1E7	76E-4	are real;
999	BIN 101	HEX F7	are integer;
“A”	”	”	are also integer;
0.493 B1	17.6B-5	1E10B-36	are fraction.

The following examples of numbers do not comply with the requirements of this standard:

27	E9	.001
“£”	”	3B - 4

7.1.4 Strings

NOTE 1 A string provides a convenient way of representing a set of byte constants such as a byte array parameter or the initial value of a byte array (see 7.2.6.3, 7.2.6.4, 7.3.5 and 7.4.10).

A string shall be basically a sequence of stringchars enclosed in double quotes (”) and shall denote the set of values formed by taking the decimal values of those characters as specified in Table 2. As a consequence of this requirement, space characters may occur in strings (to stand for themselves) whereas newline and tab shall not occur.

The character #, which is not a stringchar, shall have a special significance; a sequence enclosed within a pair of # characters within a string (a stringinsert) shall be interpreted as part of an array initial value as specified in 7.2.6.4. Thus it shall consist of a sequence of integer constants in the range 0 to 255 (possibly followed by replication factors) separated by commas.

NOTE 2 Such sequences behave like normal program text and so newline, tab and space characters, comments and LET replacements (see 7.1.11) may occur in the normal way.

The syntax shall be:

```
string      ::= " [ stringchar ... | stringinsert ] ..."
stringinsert ::= # [ stringitem [, stringitem ]... ] #
stringitem  ::= integer [ ( length ) ]
```

NOTE 3 A string that is too long to fit on one line may be spread over several lines by using stringinserts containing newline characters. There shall be no direct way of representing a newline in a string. It shall be treated like any other character that is not a stringchar and shall be inserted in a # sequence, using the LET facility for clarity if so desired (see 7.1.11).

If the characters \$ or £ are used as alternatives to #, they shall occur as matched pairs.

Assuming that

```
LET N L = 10;
```

```
LET TAB = 9;
```

have been set, valid examples of "string" are:

"THIS IS A STRING"	is a simple example;
" # NL #"	is a single newline;
"DAY # TAB # MONTH # TAB # YEAR"	inserting tabs;
" # NL(4), TAB # ALARM"	four newlines, tab and "ALARM";
" "	a null string.

The following examples do not comply with the requirements of this standard:

```
"PRICE=£"
""""
```

7.1.5 Comments. All characters starting from and including the character % up to and including the next % character shall be treated as comment and ignored by the compiler. The layout character newline shall not be used in comments; this requirement prevents subsequent lines of program text being treated as comment if a % character is inadvertently omitted.

NOTE The layout characters space and tab may occur in comments.

The syntax shall be:

```
comment    ::= % sequence-of-RTL/2-characters-excluding-%-and-newline %
```

An example is:

```
% THIS IS A COMMENT %
```

7.1.6 Titles. A title shall provide a means of labelling the object code or a listing of the source code in whole or in part. The details shall depend upon the implementation. A title shall consist of the keyword TITLE followed by any sequence of characters not including a semicolon (;).

The syntax shall be:

```
title      ::= TITLE sequence-of-RTL/2-characters-excluding-semicolon
```

An example is:

```
TITLE DEBUG ROUTINE 13 MAY 1980
```

7.1.7 Options. The provision of an option statement is deprecated.

NOTE For compatibility with option statements included prior to the implementation of amendment No. 2, the keyword "OPTION" remains reserved.