Appendices

TABLE OF CONTENTS

Appendix A: MRL Format Description

Appendix B: Reserved Words/Standard Identifiers

Appendix C: ASCII Character Set

Appendix D: Language Description

Appendix E: Bibliography

Appendix F: Error Messages

MRL FORMAT DESCRIPTION

Chapter 1. General Description

The MRL format is byte oriented. It consists of a number of link items. Each link item contains a number of information bytes.

An MRL file is a sequence of link items with their associated information bytes. It bears close resemblance to a Microsoft REL file, but is byte oriented to allow a linker that is written in a high level language to operate at reasonable speed.

Some REL items don't have a MRL relative. Read the Assembly Language Interfacing Section in the **Advanced Programming Guide** for more information about how to write a translatable assembler program and how to do the actual translation.

Chapter 2. The MRL Items

16 MRL items are defined. They are listed below in order of appearance in the definition of the MrlItem enumeration type.

'card' means 2 byte representation of a cardinal number, **'name'** means ASCII representation of a name delimited by a 0C byte. They are the MRL equivalents of the REL 'A' and 'B' fields. Whereas the REL 'B' field is limited to at most 8 characters, an MRL name field is theoretically unlimited; however, ML the MRL linker allows for 24 characters at most. Longer names will be clipped.

Item Name		Usage
EndFile	card	Marks end of MRL file, 'card' is just junk.
Offset	card	Contrary to the REL format, this item stands just before the head of an ChnExt MRL Item. Its 'card' value is added to the value of the chain symbol. All items in the chain are replaced by the resulting value.

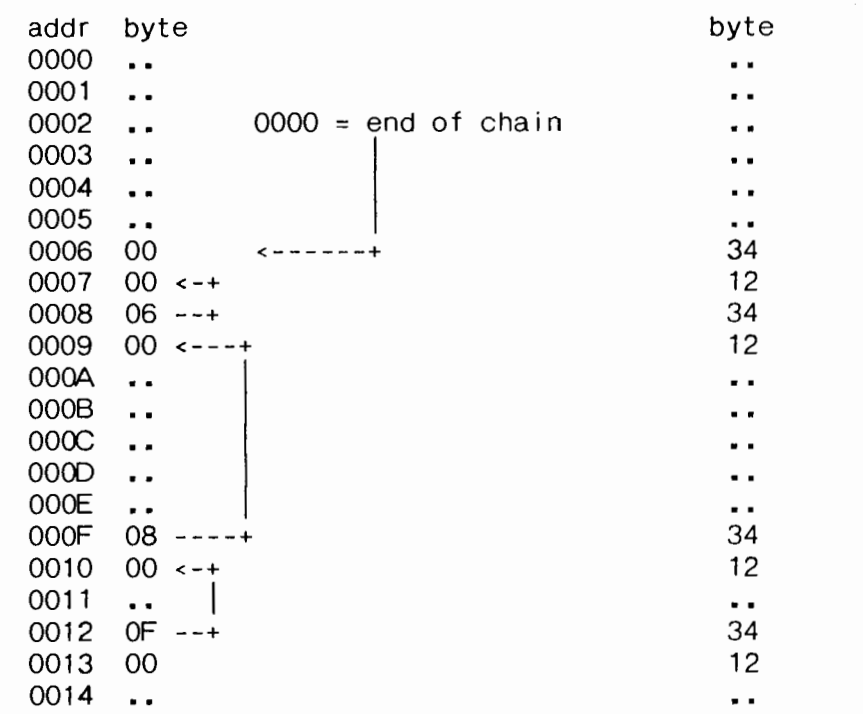
Item Name		Usage
Dword	card	Card is a data relative word, add start of module data segment to 'card' and generate the resulting word in COM file.
Cword	card	Code relative word. Add the start of the current modules code segment to 'card' before loading.
Datasiz	card	Size of data segment in bytes.
Codesiz	card	Size of code segment in bytes.
EndMod	card	End of module. If 'card' is <> 0FFFFH then 'card' is the offset of the initialization code start in the code segment. The linker will generate a call to this address before starting the main program.
ChnLoc	card	Chain the current loading location address. 'card' is the address of the chain's head. See description of chains below.
Dpublic	card name	Export item 'name', add the current module's data segment starting address to 'card' , the resulting value is the symbol's value.
Cpublic	card name	Export code relative item. 'name' is the procedure/module's name, 'card' is offset into the current global module's code segment.
Apublic	card name	Export item, value is 'card'. This item is not generated by the compiler, but can be used by assembler modules to define absolute variables. Be careful, the item hasn't been tested thoroughly.
ChnExt	card name	Chain external 'name', chain head is at offset 'card'. See chain description below.

Item Name		Usage
ModName	name	Define module name
LibReq	name	Search file 'name.MRL' for unresolved items. Each module requests all modules it imports, so user only has to give the linker a main name and the ML linker will then load all required files. The linker recognizes multiple requests for the same file and loads it once only.
DefPub	name	Informs linker that this module defines the object 'name'. Item is at beginning of file. Used when searching library files. The actual value is given later in the file by one of the 'public' items.
AbsByte	byte...byte	Load the next bytes. AbsByte **does not fit into the normal enumeration type** numbering. It is represented by a byte > 80H, the low 7 bits determine the number of absolute bytes following.

Chapter 3. Chains

Chains are characterized by their head (starting location) and the fixup value. Each element of the chain points to the next element. A value of 0 ends the chain. Each element is to be replaced by the fixup value. For the **ChnLoc** item this value is the current value of the loading location counter 'loc', for the **ChnExt** item it is the value of the indicated external.

To process a chain you recover the value presently in the chain head location, and patch the fixup value into that location. The old value of the head is the address of the next location to patch. The chain ends when the head address is 0000 (do not patch location 0000).



memory before chain processing

fixups done, chain processed

chain head = 0012H
fixup value = 1234H

The first item in an MRL file is the module name, followed by optional DefPub items (these are only needed for MODLIB file). Next comes the DataSiz item. All other items may appear in any order, with the EndMod item marking the end of the module. Several modules may be in the same file (library files built with LIB80 out of separate REL files). The EndFile item is the last item in the file.

Chapter 4. Byte Stream Representation of an MRL Link Item

```
ItemType =
          ( EndFile, Dword, Cword, Datasiz, Codesiz, EndMod, ChnLoc,
          Dpublic, Cpublic, Apublic, ChnExt, ModName, LibReq, DefPub )


bb = ORD(Item)
cl = low byte of cardinal value
ch = high byte      "
n1..nn = characters of name in ASCII
qq = 128 + number of absolute bytes following (maximum 127)


bb cl ch                    item with only a card field
bb cl ch n1 n2 .. nn 00     item with card & name field
bb n1 n2 .. nn 00           item with only a name filed
qq n1 n2 .. nn              AbsByte with n bytes following
```

The sample assembly module Silly (See **Advanced Programming Guide**) is presented here in its MRL byte stream representation.

MRL hex Bytes	MRL Item Description
0C 53 69 6C 6C 79 00	ModName Silly
0E 53 69 6C 6C 79 2E 45 71 54 65 73 74 00	DefPub Silly.EqTest
0E 53 69 6C 6C 79 00	DefPub Silly
04 01 00	DataSiz 1
05 10 00	CodeSiz 10
8C FD E1 E1 7D E1 BD 3E 00 20 01 3C 32	12 Absolute bytes
02 00 00	Dword 0
82 FD E9	2 Absolute bytes
09 00 00 53 69 6C 6C 79 2E 45 71 54 65 73 74 00	Cpublic Silly.EqTest value 0
08 00 00 53 69 6C 6C 79 00	Dpublic Silly value 0
06 FF FF	End of module; no init code.
00 00 00	End of file

RESERVED WORDS AND SYMBOLS

NOTE - All reserved words are always uppercase, as Modula-2 is case sensitive. Also, you aren't allowed to abbreviate the longer ones - every letter is significant.

+	=	AND	FOR	QUALIFED
-	#	ARRAY	FROM	RECORD
*	<	BEGIN	IF	REPEAT
/	>	BY	IMPLEMENTATION	RETURN
:=	<>	CASE	IMPORT	SET
&	<=	CONST	IN	THEN
.	>=	DEFINITION	LOOP	TO
,	..	DIV	MOD	TYPE
;	:	DO	MODULE	UNTIL
(	)	ELSE	NOT	VAR
[	]	ELSIF	OF	WHILE
{	}	END	OR	WITH
^	|	EXIT	POINTER	
		EXPORT	PROCEDURE	

STANDARD IDENTIFIERS

ABS	EXCL	MIN
BITSET	FALSE	NEW
BOOLEAN	FLOAT	NIL
CAP	HALT	ODD
CARDINAL	HIGH	ORD
CHAR	INC	PROC
CHR	INCL	REAL
DEC	INTEGER	TRUE
DISPOSE	MAX	TRUNC
		VAL

ASCII CHARACTER SET

0	000	00	nul	32	040	20		64	100	40	@	96	140	60	`
1	001	01	soh	33	041	21	!	65	101	41	A	97	141	61	a
2	002	02	stx	34	042	22	"	66	102	42	B	98	142	62	b
3	003	03	etx	35	043	23	#	67	103	43	C	99	143	63	c
4	004	04	eot	36	044	24	$	68	104	44	D	100	144	64	d
5	005	05	enq	37	045	25	%	69	105	45	E	101	145	65	e
6	006	06	ack	38	046	26	&	70	106	46	F	102	146	66	f
7	007	07	bel	39	047	27	'	71	107	47	G	103	147	67	g
8	010	08	bs	40	050	28	(	72	110	48	H	104	150	68	h
9	011	09	ht	41	051	29	)	73	111	49	I	105	151	69	i
10	012	0A	lf	42	052	2A	*	74	112	4A	J	106	152	6A	j
11	013	0B	vt	43	053	2B	+	75	113	4B	K	107	153	6B	k
12	014	0C	ff	44	054	2C	,	76	114	4C	L	108	154	6C	l
13	015	0D	cr	45	055	2D	-	77	115	4D	M	109	155	6D	m
14	016	0E	so	46	056	2E	.	78	116	4E	N	110	156	6E	n
15	017	0F	si	47	057	2F	/	79	117	4F	O	111	157	6F	o
16	020	10	dle	48	060	30	0	80	120	50	P	112	160	70	p
17	021	11	dc1	49	061	31	1	81	121	51	Q	113	161	71	q
18	022	12	dc2	50	062	32	2	82	122	52	R	114	162	72	r
19	023	13	dc3	51	063	33	3	83	123	53	S	115	163	73	s
20	024	14	dc4	52	064	34	4	84	124	54	T	116	164	74	t
21	025	15	nak	53	065	35	5	85	125	55	U	117	165	75	u
22	026	16	syn	54	066	36	6	86	126	56	V	118	166	76	v
23	027	17	etb	55	067	37	7	87	127	57	W	119	167	77	w
24	030	18	can	56	070	38	8	88	130	58	X	120	170	78	x
25	031	19	em	57	071	39	9	89	131	59	Y	121	171	79	y
26	032	1A	sub	58	072	3A	:	90	132	5A	Z	122	172	7A	z
27	033	1B	esc	59	073	3B	;	91	133	5B	[	123	173	7B	{
28	034	1C	fs	60	074	3C	<	92	134	5C	\	124	174	7C	|
29	035	1D	gs	61	075	3D	=	93	135	5D	]	125	175	7D	}
30	036	1E	rs	62	076	3E	>	94	136	5E	^	126	176	7E	
31	037	1F	us	63	077	3F	?	95	137	5F	_	127	177	7F	del

LANGUAGE DEFINITION

This appendix lists the subset of Modula-2 implemented currently by the Modula-2 System for Z80 CP/M.

```
Character = " " | .. | 176C .
Letter = "A" | .. | "Z" | "a" | .. | "z" .
Ident = Letter { Letter | Digit } .
Number = Integer .
Integer = Digit { Digit } |
          OctalDigit { OctalDigit } ( "B" | "C" ) |
          Digit { HexDigit } "H" .
HexDigit = Digit | "A" | "B" | "C" | "D" | "E" | "F" .
Digit = OctalDigit | "8" | "9" .
OctalDigit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" .
Real = Sign Number "." [ Number ] [ "E" Sign Number ] .
Sign = [ "+" | "-" ] .
String = " ' " { Character } " ' " | ' " ' { Character } ' " ' .
Qualident = Ident { "." Ident } .

ConstantDeclaration = Ident "=" ConstExpression | Ident "=" ( String | Real ).
ConstExpression = SimpleConstExpr [ Relation SimpleConstExpr ] .
Relation = "=" | "#" | "<>" |"<" | "<=" | ">" | "=>" | IN .
SimpleConstExpr = Sign ConstTerm { AddOperator ConstTerm } .
AddOperator = "+" | "-" | OR .
ConstTerm = ConstFactor { MulOperator ConstFactor } .
MulOperator = "*" | "/" | DIV | MOD | AND | "&" .
ConstFactor = Qualident | Number | String | Set | "(" ConstExpression ")" |
     NOT ConstFactor .
Set = [ Qualident ] "{" [ Element { "," Element } ] "}" .
Element = ConstExpression [ ".." ConstExpression ] .
```

```
TypeDeclaration = Ident "=" Type .
Type = SimpleType | ArrayType | RecordType | SetType | PointerType .
SimpleType = Qualident | Enumeration | SubrangeType .
Enumeration = "(" IdentList ")" .
IdentList = Ident { "," Ident } .
SubrangeType = "[" ConstExpression ".." ConstExpression "]" .
ArrayType = ARRAY SimpleType { "," SimpleType } OF Type .
RecordType = RECORD FieldListSequence END .
FieldListSequence = FieldList { ";" FieldList } .
FieldList = [ IdentList ":" Type |
     CASE [ Ident ":" ] Qualident OF Variant { "|" Variant }
     [ ELSE FieldListSequence ] END ] .
Variant = CaseLabelList ":" FieldListSequence .
CaseLabelList = CaseLabels { "," CaseLabels } .
CaseLabels = ConstExpression [ ".." ConstExpression ] .
SetType = SET OF SimpleType .
PointerType = POINTER TO Type .
FormalTypeList = "(" [ [ VAR ] FormalType
     { "," [ VAR ] FormalType } ] ")" [ ":" Qualident ] .

VariableDeclaration = VariableIdentList ":" Type .
VariableIdentList = VariableDesignator { "," VariableDesignator } .
VariableDesignator = Ident [ "[" Number "]" ] .

Designator = Qualident { "." Ident | "[" ExpList "]" | "^" } .
ExpList = Expression { "," Expression } .
Expression = SimpleExpression [ Relation SimpleExpression ] .
SimpleExpression = [ "+" | "-" ] Term { AddOperator Term } .
Term = Factor { MulOperator Factor } .
Factor = Number | String | Set | Designator [ ActualParameter ] |
     "(" Expression ")" | NOT Factor .
ActualParameter = "(" [ ExpList ] ")" .
```

```
Statement = [ Assignment | ProcedureCall | IfStatement | CaseStatement |
     WhileStatement | RepeatStatement | LoopStatement | ForStatement |
     WithStatement | EXIT | RETURN [ Expression ] ] .
Assignment = Designator ":=" Expression .
ProcedureCall = Designator [ ActualParameter ] .
StatementSequence = Statement { ";" Statement } .
IfStatement = IF Expression THEN StatementSequence
     { ELSIF Expression THEN StatementSequence }
     [ ELSE StatementSequence ] END .
CaseStatement = CASE Expression OF Case { "|" Case }
     [ ELSE StatementSequence ] END .
Case = CaseLabelList ":" StatementSequence .
WhileStatement = WHILE Expression DO StatementSequence END .
RepeatStatement = REPEAT StatementSequence UNTIL Expression .
ForStatement = FOR Ident ":=" Expression TO Expression
     [ BY Number ] DO StatementSequence END .
LoopStatement = LOOP StatementSequence END .
WithStatement = WITH Designator DO StatementSequence END .

ProcedureDeclaration = ProcedureHeading ";" Block Ident .
ProcedureHeading = PROCEDURE Ident [ FormalParameter ] .
Block = { Declaration } [ BEGIN StatementSequence ] END .
Declaration = CONST { ConstantDeclaration ";" } |
     TYPE { TypeDeclaration ";" } | VAR { VariableDeclaration ";" } |
     ProcedureDeclaration ";" | ModuleDeclaration ";" .
FormalParameter = "(" [ FPSection { ";" FPSection } ] ")" [ ":" Qualident ] .
FPSection = [ VAR ] IdentList ":" FormalType .
FormalType = Qualident .

ModuleDeclaration = MODULE Ident ";" { Import } [ Export ] Block Ident .
Export = EXPORT [ QUALIFIED ] IdentList ";" .
Import = [ FROM Ident ] IMPORT IdentList ";" .
DefinitionModule = DEFINITION MODULE Ident ";" { Import }
     [ Export ] { Definition } END Ident "." .
Definition = CONST { ConstantDeclaration ";" } |
     TYPE { Ident [ "=" Type ] ";" } | VAR { VariableDeclaration ";" } |
     ProcedureHeading ";" .
ProgramModule = MODULE Ident ";" { Import } Block Ident "." .
CompilationUnit = DefinitionModule | [ IMPLEMENTATION ] ProgramModule .
```

In this subset, Procedure Types and Open Array Parameters are missing.

BIBLIOGRAPHY

This bibliography cannot be complete. If you have the time, go to a good technical bookstore and look for other Modula-2 related books. Because Modula-2 is catching on as a language, new books are published at an astounding rate.

N. Wirth: Programming in Modula-2, Springer, 1983: Although most reviewers aren't pleased with the ill-organized index and the overall organization of the book, this is **the** standard Modula-2 book simply because it defines the Modula-2 language standard.

Dal Cin, Lutz, Risse: Programmierung in Modula-2, Teubner Studienskripten, Stuttgart 1984: This book is written in **german.** It is not simply a translation of the standard text above, but describes a UNIX implementation as well as the concepts present in Modula-2. This book is relatively easy to read.

Remmele, Schecher, Eds.: Microcomputing, Teubner Stuttgart 1979: This book is a collection of small articles not only covering Modula-2 and Lilith, which are partially written in german.

G. Pomberger: Softwaretechnik und Modula-2, Oldenbourg Verlag, 1984: Mr. Pomberger had a longer stay at the Institut fuer Informatik, and wrote this book (in **german**) with the aid of Lilith. This seems to be quite a good text about software engineering generally. Modula-2 was the language chosen for the approach.

R. Gleaves: Modula-2 for Pascal Programmers, Springer-Verlag, New York, 1984: This book received very favorable comments in several reviews. This is no surprise since Richard Gleaves is the author of the very well written Volition Systems Manual. Although not a cheap buy at about 10 cents a page, this book is worth its money. Here, the syntax graphs missing in **Programming in Modula-2**, are presented. Unfortunately, Richard Gleaves didn't use the MODUS Standard and Utility Library, but a revised version of Volition's, which is close to the first but not equal.

There are several **Reports of the Institut fuer Informatik** at ETH, Zuerich that treat several interesting aspects of programming with Modula-2 and Lilith. Perhaps you can get them in University Libraries.

A lot of Modula-2 programming is done in Australia, mostly on UNIX systems. There should be literature from that part of the earth, too.

ERROR MESSAGES

Chapter 1. Compiler Error Messages

The compiler's error messages are explained in detail below. A shorthand list to copy and hold at hand can be found at the end of this appendix.

If a message is self explanatory, no additional text at all is appended to it.

Section 1. Errors in Pass 1

0: illegal character in source file

control characters in the source are most often the causes of this error message.

2: constant out of range

may also be issued by Pass 2.

3: open comment at end of file

4: string terminator not on this line

string constants may not exceed a source line; the start and termination character must be the same (either both double quotes or both single quotes).

5: too many errors

after 300 detected errors, this message is output.

6: string too long

the maximum string length is 128 bytes. This is a limit imposed by the compiler's internal string buffer.

7: to many identifiers (name table full)

Shorten your identifiers! The name table holds at most 5120 characters.

20: identifier expected
21: integer constant expected
22: ']' expected
23: ';' expected
24: block name at the END does not match
25: error in block
26: ':=' expected
27: error in expression
28: THEN expected
29: error in LOOP statement
30: constant must not be CARDINAL
31: error in REPEAT statement
32: UNTIL expected
33: error in WHILE statement
34: DO expected
35: error in CASE statement
36: OF expected
37: ':' expected
38: BEGIN expected
39: error in WITH statement
40: END expected
41: ')' expected
42: error in constant
43: '=' expected
44: error in TYPE declaration
45: '(' expected
46: MODULE expected
47: QUALIFIED expected
48: error in factor
49: error in simple type
50: ',' expected
51: error in formal type
52: error in statement sequence
53: '.' expected

54: export at global level not allowed

use qualified export (EXPORT QUALIFIED), if you aren't trying to export from a program module.

55: module body in definition module not allowed
56: TO expected
57: nested (local) module in definition module not allowed
58: '}' expected
59: '..' expected

60: error in FOR statement
61: IMPORT expected

Section 2. Errors in Pass 2 and 3

70: identifier specified twice in importlist

eventually, you imported an enumeration constant and its associated type which automatically re-imports this constant.

71: identifier not exported from qualifying module
72: identifier declared twice
73: identifier not declared
74: type not declared

75: identifier already declared in module environment

if you cannot find such an identifier, maybe it is an enumeration constant

77: too many nesting levels

the compiler allows for 16 nesting levels including WITH statement levels.

78: value of absolute address must be of type CARDINAL

79: scope table overflow in compiler

see error 77.

81: definition module belonging to implementation not found
82: structure not allowed for implementation of hidden type
83: procedure implementation different from definition
84: not all defined procedures or hidden types implemented

watch for case errors in identifiers! This error is detected at the end of the implementation module.

85: name conflict of exported object or enumeration constant in environment

86: incompatible version of symbolic modules

this error is used for version control which is currently disabled.

88: function type is not scalar or basic type
90: pointer-referenced type not declared
91: tagfield type expected
92: incompatible type of variant-constant
93: constant used twice

94: arithmetic error in evaluation of constant expression

an over- or underflow occured during evaluation of a constant expression. Try to set the expression up in a way that no such condition can occur.

95: incorrect range

ranges have to be of the form lowBound..hiBound.

96: range only with scalar type
97: type-incompatible constructor element

98: element value out of bound

set element ordinal values have to be between 0 and 15. No sets of, for instance, 100..115 are possible.

99: set-type identifier expected

set constants have to be preceded by their type identifier if they aren't BITSETs.

100: structured type too large

size of structures are limited to 32k bytes.

101: undeclared identifier in export-list of module
102: range not belonging to base type

103: wrong class of identifier

identifier classes are: constants, types, variables, procedures and modules. Example: you used a variable instead of a type or a constant.

104: no such module name found

perhaps a module identifier case problem. (i.e. you import from Term1 but the module is actually called TERM1. The compiler finds, in that case, a matching file named TERM1.MSY, but the difference in module names cannot be detected until Pass 2 (in definition modules) or Pass 3 (in implementation modules) terminates.

105: modulename expected
107: set too large

sets may consist of 16 elements at most.

109: scalar or subrange type expected

110: case labels out of bounds

case labels supersede subrange bounds, for example.

111: illegal export from program module

120: incompatible types in conversion

for 'wild' type transfers, at least the size of the converted objects has to coincide. Only CHR, ORD and VAL can convert different sized objects into each other. Furthermore, both operands have to be either of (scalar, set, or pointer) type or of (record or array) type. This is an implementation restriction.

121: this type is not expected

122: variable expected

often set instead of an error 73, undeclared identifer.

123: incorrect constant
126: set constant out of range
127: error in standard procedure parameters
128: type incompatibility
129: type identifier expected
130: type impossible to index
131: field not belonging to a record variable
132: too many parameters
134: reference not to a variable
135: illegal parameter substitution
136: constant expected

137: expected parameter(s)

the number of parameters you gave does not match the number of parameters declared.

138: BOOLEAN type expected
139: scalar type expected
140: operation with incompatible type
141: only global procedure allowed as procedure variable value
142: incompatible element type
143: type incompatible operands
144: no selector allowed for procedures
145: only function calls allowed in expression
146: arrow not belonging to a pointer variable

147: standard function or procedure must not be assigned

because they have different parameter passing mechanisms, you cannot use standard functions/procedures as variables. Applies only to Procedure Types.

148: constant not allowed as variant
149: SET type expected

150: illegal substitution to WORD parameter

any two byte scalar can be substituted for WORD.

151: EXIT only in LOOP
152: RETURN only in PROCEDURE
153: expression expected
154: expression not allowed
155: type of function expected

156: integer constant expected

a BY clause has to have a constant value of type INTEGER or CARDINAL.

157: procedure call expected
158: identifier not exported from qualifying module

223: case label twice specified

the label can be specified twice explicitly or one time as part of a label range.

Section 3. Additional Errors for Pass 1

331: no priority allowed
337: more than 16 LOOP-Statements nested

these are all implementation restrictions.

338: different identifier with same significant base

the compiler distincts identifiers but in the first 14 places. This was done because of the linker's 24 character maximum identifier length. An identifier used by the linker consists of ModuleName.ObjectName, i.e. it is a qualified identifier.

360: illegal definition
361: illegal declaration
362: illegal block identifier

364: filename incompatible with modulename

a module's file name is formed by uppercasing the first 8 characters of the module name. If it is shorter than eight characters, the rest is padded with blanks. There is NO exception to this rule.

365: filename extension incompatible with module type

use DEF for definition modules only. MOD applies to (IMPLEMENTATION) MODULEs.

366: illegal position of BEGIN

back in Pascal, aren't you?

367: hidden type only allowed in DEFINITION MODULE

may occur also if a colon (':') instead of an equal sign ('=') is set in a type definition.

Section 4. Additional Errors for Pass 3

401: too many case labels (more than 256).

this is an implementation restriction.

402: overlapping case ranges

you specified ranges of the form 10..20 and 15..25; eventually it is only a single label that lies inmidst a range also specified.

403: expression too complicated

this error indicates that an overflow of the expression evaluation buffer of Pass 3 occured. Because this error is fatal, the compilation doesn't continue. Try to simplify the constant expression that is involved.

Section 5. Additional Errors for Pass 4

500: type conflict in expression
501: unexpected token from interpass file
502: structured function return value not allowed
503: name in factor is not constant, variable or function
504: array or record constant not allowed
505: illegal standard name in procedure/function
506: illegal standard name in constant
507: illegal type conversion
508: module name not exported
509: more than 16 LOOP statements nested
510: loop stack overflow
511: name is not variable in procedure: AccessVariable
512: name is not constant in procedure: Constant
513: type is not array in procedure: IndexVar
514: type is not record in procedure: FieldVar
515: name is not field in procedure: FieldVar
516: type is not procedure in procedure: PointVar
517: name is not variable in procedure: Variable
518: peephole table overflow at initialisation (> 50 entries)
519: constant expression stack overflow (> 16 entries)
520: constant expression stack underflow
524: illegal M-code detected in module McExpnd
525: M-Code BAD detected (usually means illegal operation for specified operand type)
526: M-Code not yet implemented
527: name table overflow (> 5120 characters)
528: spelling index out of range in procedure GetSpelling
530: string too long (> 128)
531: illegal library routine in procedure EmitSpelling
532: illegal link item in procedure EmitItem
533: label table overflow (> 700 labels)
534: external table overflow (> 200 entries)
535: type is not array in procedure ParameterList
540: expression passed to ARRAY OF WORD
541: internal error concerning Open Array passed as Open Array
542: unexpected kind of variable
543: non-global procedure address assignment.

Most of these messages indicate a compiler bug. Please notify us about the problem by sending us a preferably short listing of a program that provokes the error.

Section 6. Other Errors

Besides these numbered errors, there are some more error messages related to files and command lines, as well as to internal error conditions.

1. Internal Errors

---- COMPILER ABORTED: Lookahead Too Long at Line xxx

Pass 1 has to decide whether a given statement is an assignment or a procedure call. This is done by using a lookahead feature. If your statement is too complicated (i.e. has too many selectors), Pass 1 can't keep the necessary information in its buffer. Use a WITH statement to shorten the number of selectors.

---- INTERNAL ERROR

Unspecified internal error.

---- Too Many Errors

more than 300 errors occured.

---- Compilation Aborted by ^C

you entered ^C at the keyboard.

2. Compiler-Parts Related Errors

---- Version Conflict

different parts of the compiler belong to different versions of the system. Copy your compiler again **completely** from the newest master disk(s).

3. Commandline Related Errors

---- Illegal Command Line

general command line error.

---- Switch Specifier Expected

you set a '/' without a switch following it.

---- Illegal <switch>-Switch

<switch> does not exist, or you used incorrect arguments.

---- Illegal Module Type

module (= file) type can be but MOD or DEF. MOD is the default file type.

4. File I/O Related Errors

---- Cannot Open File "filename.typ"

---- Cannot Close File "filename.typ"

This eventually indicates a full directory.

---- Cannot Create File "filename.typ"

The directory has to be full.

---- Cannot Write to File "filename.typ"

---- Cannot Read From File "filename.typ"

may be either an empty or scrambled file.

---- SUBMIT Aborted

displayed if errors occured and the X switch was specified.

Chapter 2. Linker Error Messages

Normally, linker errors occur only if incorrect assembly language modules are used or if you made a version mix between definition modules and their associated implementation modules. All error messages are given literally by the linker. There is one non fatal error message:

---- Circular Reference Detected - FileName FileName {FileName}

A circular reference can be constructed as follows:

- You have two modules, say M1 and M2.
- M1.MOD imports some items from M2.DEF.
- M2.MOD imports some items from M1.DEF.

That's it. This scheme can be extended to include three and more modules. Circular references prevent the linker from creating an 'absolutely correct' initialization order. This in turn means that you shouldn't use any items of the circular referenced modules in one of their bodies (i.e. initializations).

All other error messages are fatal and lead to no output file at all. They are:

---- <symbol> Twice Declared

A symbol has been defined twice in a single module.

---- <symbol> Not Defined

Undefined symbols can be provoked by compiling a definition module, then its implementation and then again the definition module with some more procedures in it, forgetting the implementation.

---- No Starting Address Defined

You cannot use an assembler module as your main program; it is impossible to set the starting address within an assembler module.

---- Negative Load Index (Bad Assembler Module?)

Perhaps you used an assembler module which wasn't correctly converted.

---- Module Too Big

The maximum module size accepted by the linker is about 25 kBytes. Our observations lead to the conclusion that this would be a module of about 4000 lines. Don't you think that this is a little bit too big to be put into one single module?

---- Program is > 64k

See linker description for some advice...

---- Overlapping Code & Data Segments

Using the /C and / or /D switch of the linker, you overlaid code and data. Look at the statistics that are output in verbose mode, and correct the /C and /D values appropriately.

---- Overlapping Code & Shared Segments

The /H switch specifies the end of the shared area. You have to leave at least the size of the shared module's data between end of code and start of heap addresses to have enough space.

---- Overlapping Data & Heap Segments

The only thing to do is to enlarge the /H switch's value.

---- No Stack-Heap Space

/T and /H cannot have the same value; also the /D value plus the data size has to be smaller than /H or /T, respectively.

---- Illegal Format of Shared Module

Your shared module contains code. You cannot use modules that contain code (i.e. a module body or procedures) as shared data module.

---- Use of S-Switch Without H-Switch

To use the linker's S switch, it is mandatory to set the H switch, too, because it defines the shared data module's end address. (See also **Anatomy of a Modula-2 Program** in the linker description of the **Implementation Guide**).

---- J-Switch < 3

Because the J switch generates a jump around a reserved area at the program start, you need to specify at least 3 bytes of reserved space to allow the linker to generate a jump. A jump machine instruction sequence is 3 bytes long.

---- Unexpected EOF

This error message appears if one of your MRL files has been cut. Find the file and copy it again from another disk resp. recompile the module.

---- Errors in Pass 1 ----

If another error message was issued by Pass 1 of the linker, this message is written at the end of it.

---- Errors in Pass 2 ----

If another error message was issued by Pass 2 of the linker, this message is written at the end of this Pass.

---- SUBMIT Aborted

If the X-switch has been specified and error(s) occured, this message confirms the interruption of the submit process.

---- Linker Aborted by ^C

This message is displayed if you entered ^C at the keyboard during the link process.

---- Version Conflict

The linker internally checks if ML.COM and MLP2.COM belong to the same linker version. If these versions are different, the link pocess is aborted.

---- Cannot Find MLP2.COM

Pass 2 couldn't be found on the drives indicated by the program search path installed into the linker.

Chapter 3. Runtime Error Messages

There are only very few runtime errors generated by the Modula-2 System. The 'System' consists, in this respect, of the MODLIB routines. Other modules (from the libraries) may generate more messages. See their detailed description for this messages. The System messages are:

---- Modula-2 Runtime Error: Out Of Memory

The abort condition

```
TopOfStack - HeapLimit >= 256
```

is tested on every procedure entry. This test cannot be disabled. If you need a special version of MODLIB not doing this test because you want to put your code into ROM, please contact us.

Another error message is

---- Modula-2 Runtime Error: Cannot Execute Chain File

This error message is issued if you try to start a program that was linked using the /N switch of the linker, omitting the heap initialization. If you want to build a system of programs chaining each other and sharing data, the first one has to initialize the heap.

Should a REAL calculation lead to an overflow, the program gets immediately aborted and displays the message

---- Modula-2 Runtime Error: REAL Overflow

A fourth message shouldn't get issued at all. There is some really serious error in your program if it appears. It is:

---- Modula-2 Runtime Error: Unknown

These were all the runtime error messages issued by the system.

Chapter 4. MR Error Messages

The error messages emitted by MR are fatal but do not interrupt the program. If an error occurs, the resulting converted file, though, is not correct and therefore not usable. These are the error messages:

---- error in conversion file

'conversion file' means the name translation table. Your R2M typed file does not conform to the specifications given in the **Advanced Programming Guide.**

---- twice used in conversion file

You used the given symbol twice in the name translation file. You cannot assign the same name to two different symbols.

---- cannot load bytes outside code segment

As mentioned earlier, initialized data areas aren't supported by the ML linker.

---- cannot convert Common item

Common stinks of FORTRAN; no self respecting Modula-2 system would accept such an item ...

---- cannot convert Extension item

Your REL file contains one of the two 'reserved for future expansion' items (See Microsoft REL format description in the **L80 linker manual** by Microsoft).

---- chain address outside of code segment

There seems to be a corrupted file or an incorrectly converted one around.

---- illegal item returned from getrel procedure

This indicates that you have initialized data in your REL file.

---- DS not allowed in code segment, use DB

As stated earlier, no DS (and no ORG that sets the loading counter back) statement is supported in the code segment.

---- external +/- offset not allowed

Because of different construction of chains, external plus/minus offset cannot be converted easily from REL to MRL although both formats have external plus/minus offset constructs. Sorry, you have to do this offsetting in your code.

Chapter 5. Compiler Error Messages Shorthand

These last pages are laid out to be copied to have them at hand when working with the system.

0: illegal character in source file

2: constant out of range
3: open comment at end of file
4: string terminator not on this line
5: too many errors
6: string too long
7: to many identifiers (identifier table full)

20: identifier expected
21: integer constant expected
22: ']' expected
23: ';' expected
24: block name at the END does not match
25: error in block
26: ':=' expected
27: error in expression
28: THEN expected
29: error in LOOP statement
30: constant must not be CARDINAL
31: error in REPEAT statement
32: UNTIL expected
33: error in WHILE statement
34: DO expected
35: error in CASE statement
36: OF expected
37: ':' expected
38: BEGIN expected
39: error in WITH statement
40: END expected
41: ')' expected
42: error in constant
43: '=' expected
44: error in TYPE declaration
45: '(' expected
46: MODULE expected
47: QUALIFIED expected
48: error in factor
49: error in simple type
50: ',' expected
51: error in formal type

52: error in statement sequence
53: '.' expected
54: export at global level not allowed
55: module body in definition module not allowed
56: TO expected
57: nested module in definition module not allowed
58: '}' expected
59: '..' expected
60: error in FOR statement
61: IMPORT expected

70: identifier specified twice in importlist
71: identifier not exported from qualifying module
72: identifier declared twice
73: identifier not declared
74: type not declared
75: identifier already declared in module environment

77: too many nesting levels
78: value of absolute address must be of type CARDINAL
79: scope table overflow in compiler
81: definition module belonging to implementation not found
82: structure not allowed for implementation of hidden type
83: procedure implementation different from definition
84: not all defined procedures or hidden types implemented
85: name conflict of exported object or enumeration constant in environment
86: incompatible version of symbolic modules
88: function type is not scalar or basic type
90: pointer-referenced type not declared
91: tagfieldtype expected
92: incompatible type of variant-constant
93: constant used twice
94: arithmetic error in evaluation of constant expression
95: incorrect range
96: range only with scalar type
97: type-incompatible constructor element
98: element value out of bound
99: set-type identifier expected
100: structured type too large
101: undeclared identifier in export-list of module
102: range not belonging to base type
103: wrong class of identifier
104: no such module name found
105: modulename expected

107: set too large

109: scalar or subrange type expected
110: case labels out of bounds
111: illegal export from program module

120: incompatible types in conversion
121: this type is not expected
122: variable expected
123: incorrect constant
126: set constant out of range
127: error in standard procedure parameters
128: type incompatibility
129: type identifier expected
130: type impossible to index
131: field not belonging to a record variable
132: too many parameters

134: reference not to a variable
135: illegal parameter substitution
136: constant expected
137: expected parameter
138: BOOLEAN type expected
139: scalar type expected
140: operation with incompatible type
141: only global procedure allowed as procedure variable value
142: incompatible element type
143: type incompatible operands
144: no selector allowed for procedures
145: only function calls allowed in expression
146: arrow not belonging to a pointer variable
147: standard function or procedure must not be assigned
148: constant not allowed as variant
149: SET type expected
150: illegal substitution to WORD parameter
151: EXIT only in LOOP
152: RETURN only in PROCEDURE
153: expression expected
154: expression not allowed
155: type of function expected
156: integer constant expected
157: procedure call expected
158: identifier not exported from qualifying module

223: case label twice specified

331: no priority allowed
337: more than 16 LOOP-Statements nesteted
338: different identifier with same significant base

360: illegal definition
361: illegal declaration
362: illegal block identifier
364: filename incompatible with modulename
365: filename extension incompatible with module type
366: illegal position of BEGIN

367: hidden type only allowed in DEFINITION MODULE

401: case label range too big (maxLabel - minLabel > 256)
402: overlapping case ranges
403: expression too complicated

500: type conflict in expression
501: unexpected token from interpass file
502: structured function return value not allowed
503: name in factor is not constant, variable or function
504: array or record constant not allowed
505: illegal standard name in procedure/function
506: illegal standard name in constant
507: illegal type conversion
508: module name not exported
509: more then 16 LOOP statements nested
510: loop stack overflow
511: name is not variable in procedure: AccessVariable
512: name is not constant in procedure: Constant
513: type is not array in procedure: IndexVar
514: type is not record in procedure: FieldVar
515: name is not field in procedure: FieldVar
516: type is not procedure in procedure: PointVar
517: name is not variable in procedure: Variable
518: peephole table overflow at initialisation (> 50 entries)
519: constant expression stack overflow (> 16 entries)
520: constant expression stack underflow
524: illegal M-code detected in module McExpnd
525: M-Code BAD detected (usually means illegal operation for
specified operand type)
526: M-Code not yet implemented
527: name buffer overflow (> 5120 characters)
528: spelling index out of range in procedure GetSpelling
529: too many labels (> 32767)
530: string too long (> 128)
531: illegal library routine in procedure EmitSpelling
532: illegal link item in procedure EmitItem
533: label table overflow (> 700 labels)
534: external table overflow (> 200 entries)
535: type is not array in procedure ParameterList
540: expression passed to ARRAY OF WORD
541: internal error concerning Open Array passed as Open Array
542: unexpected kind of variable
543: non-global procedure address assignment.