C180

Common Modules Mathematical Library (CMML)

ERS

Revision E

August, 1985

1.0 PREFACE

1.1 PREFACE TO REVISION E

Revision E of the Common Modules Mathematical Library (CMML) External Reference Specification (ERS) describes CMML capabilities at Release 1.1.3. This revision incorporates features that were specified in approved DAPs and other corrections and clarifications to the text since the last complete update of the ERS.

The mathematical functions, COTAN, EXTB, and INSB (DAP S4945) are new features for Release 1.1.3. Their error numbers have been changed from the ones specified in the DAP. The VAX_to_C180 conversion routines (DAP S4821) were released at 1.1.2.

1.2 SCOPE

The C180 Mathematical Library, as defined in this document, is called the Common Modules Math Library (CMML), but is commonly referred to as MATHLIB or the Math Library. It is a collection of mathematical functions and routines, numeric and data conversion routines, and assembly language support system (ALSS) routines that provide access to some machine language instruction capabilities not otherwise available to non-assembly language programs. The numeric conversion and assembly language support routines will be referred to jointly as the CMML Common Support routines in this document.

This document gives the external specifications of the CMML but also includes some internal details because of its frequent use by product set developers. The ALSS routines formerly specified in DCS document S3410, have been incorporated here because they are now a standard part of the CMML. The CMML common support modules are discussed separately from the mathematical functions because they differ in linkage interface and error handling.

Three appendices are included. Appendix A contains the CYBIL constant and type declarations needed by the numeric-conversion and ALSS routines. Appendix B contains the error message templates used by the mathematical functions and routines. Appendix C contains a listing of the file used in converting CMML's common deck PL from MADIFY format to SCU format.

This document does not include information on the algorithms used by CMML routines or error analyses of these routines. The algorithms are in a state of flux, and the tools needed for error analyses do not currently exist. This information will be published in the CMML Reference Manual.

1.0 PREFACE
1.2 SCOPE

For performance reasons, most of the CMML routines will be written in C180 assembly language. Some of the accessory and error processing code will be written in CYBIL.

1.3 REFERENCES

+

- Cyber 180 Mainframe Model-Independent General Design Specification (MIGDS) DCS Log Id ARH1700.
- Cyber 180 System Interface Specification (SIS) DCS Log Id S2196.
- CMML Assembly-Language Support System (ALSS) DCS Log Id S3410.
- VAX File Migration DAP, DCS Log Id S4743.
- CMML VAX to C180 Conversion Routines DAP, DCS Log Id S4821.
- CMML ERS C180 Product Set and CDC FORTRAN DAP, DCS Log Id S4945.

2.0 MATHEMATICAL FUNCTIONS AND ROUTINES

2.1 INTRODUCTION

The Mathematical Routines of CMML are used to evaluate commonly occurring mathematical functions and operations, and those required by the various language standards. All mathematical routines will be written in C180 Assembly Language (exceptions to this will be specified in the IPP update).

Many of the functions of the Math Library will be implemented in-line by C180 products. The in-line version of a function returns the same result (for the same argument list) as the Math Library.

2.2 NUMBER TYPES

The mathematical routines deal with computations upon four different number types:

1. INTEGER

An integer number is a one-word right-justified two's complement 64-bit representation of a value with a magnitude in the range from -2**63 to 2**63-1.

(Reference the C180 MIGDS, section 2.2.2.)

All integers are considered standard forms.

2. SINGLE (single precision floating point)

A single precision floating point number consists of a sign bit, S, which is the sign of the fraction, a signed biased exponent (15 bits), and a fraction (48 bits) which is also called a coefficient or a mantissa. (Reference the C180 MIGDS, section 2.4.1.)
Single precision floating point (real) numbers in the C180 consist of two types, (not including coefficient sign), standard and non-standard. The standard numbers are those with exponents in the range 3000(16)..4FFF(16), inclusive, which have a non-zero fraction. Standard numbers also come in two types, normalized and unnormalized. A normalized standard number has a one in bit position 16 (i.e., the most significant bit of the fraction).

The range in magnitude, M, covered by standard, normalized single precision numbers is

```
2**-4097  <=  M  <=  (1-2**-48) * 2**4095
```


(Approximately 14.4 decimal digits of precision).

Non-standard floating point numbers have many representations;

+/-INF +/-Infinite	[S,5000000000000000(16)] Floating point numbers having exponents in the range 5000(16)..6FFF(16).
+/-IND +/-Indefinite, INDEF	[S,7000000000000000(16)] Floating point numbers having exponents in the range 7000(16)..7FFF(16).
Zero (Z1)	Zero: Floating point numbers having exponents in the range 0000(16)..0FFF(16).
Zero (Z2)	Underflow, zero: Floating point numbers having exponents in the range 1000(16)..2fff(16).
Zero (Z3)	Zero: An unnormalized floating point number with a zero fraction and a standard exponent.
Zero (0)	Zero: A sign bit followed by 63 zero bits.

(Reference the C180 MIGDS, Section 2.4.1.2-2.4.1.3 and Table 2.4-1 for a full discussion of floating point numbers.)

3. DOUBLE (precision floating point)

A double precision floating point number consists of two words, both of which are single precision numbers. The coefficient of the second word is considered to be an extension of the fraction of the first word, yielding a 96-bit fraction.

The exponent of the second word must be identical to that of the first word.

The type of the first single number determines the type of the double number.

The range in magnitude, M, covered by standard, normalized double precision numbers is

```
2**-4097  <=  M  <=  (1-2**-96) * 2**4095
```

(Approximately 28.9 decimal digits of precision).

4. COMPLEX

A complex number consists of two words, each a single precision floating point number. The first word represents the real part of the complex number, the second word represents the imaginary part.

A complex number is considered to be +/-INDEF if either the real or imaginary part is +/-INDEF. Similarly, a complex number is considered to be +/-INF if either the real or imaginary part is +/-INF.

2.3 GENERAL RULES

The following general rules apply to the use of these number forms in computational operations within the Math Library:

Rule number one:
Unless specifically documented otherwise, if a standard number of the appropriate type is employed in a computational operation, a standard number of the appropriate type will result. The documented exceptions to this cover such things as computing an answer which exceeds the limits of the standard forms, or performing a mathematically invalid operation.

Rule number two:
Unless specifically documented otherwise, if either:

a.) A non-standard number, other than zero (0), is employed in a computational operation, or

b.) The documented limits in rule number one above are exceeded, error handling (see below) will occur. The documented exceptions to this cover some cases wherein various non-standard numbers are within the domain of the function.

These two rules define the limits of CDC support in the area and also the completeness of the supporting documentation.

2.4 DOCUMENTATION CONVENTIONS

Certain conventions and definitions are observed in this document.

- Symbolic names are always delimited by blanks, and any alphabetic letters appearing therein are in upper case.

- Both ^ and two quantities separated by a comma and enclosed in parentheses denote juxtaposition and are used in referring to complex or double precision quantities.

- All values given are in decimal, unless otherwise noted. When bit configurations are listed, the radix may be listed in parentheses after the string.

- An argument list is an ordered n-tuple of arguments [X1 , ..., Xn], where X1 , ..., Xn are the arguments in order. For convenience, we identify arguments with corresponding one-member argument lists.

- The *domain* of an entry point is the collection of argument lists for which that entry point has been designed to return meaningful results without generating an error condition.

- The *range* of an entry point is the collection of results obtained by entering members of the domain into the entry point.

- Arguments of trigonometric functions and results of inverse trigonometric functions are measured in radians, unless otherwise noted.

- The symbol * denotes multiplication, / denotes division, and ** denotes exponentiation.

2.5 LINKAGE INTERFACE

The mathematical routines are functions that return a single value to the caller. Their linkage interface conforms to the SIS conventions for scalar functions whose values are of known length less than or equal to 128 bits.

Two modes of entry are provided; a call-by-reference linkage and a call-by-value linkage. Under call-by-reference, register A4 points to the actual parameter list. Under call-by-value, the successive words of the successive arguments are laid out contiguously in the X registers, beginning with X2, as described for register call functions in the SIS. For example, the calling sequence to MLP$VITOD uses registers X2, X3, X4, where X2 holds the integer base, and X3^X4 holds the double precision exponent. (This is in accordance with the SIS for C180 software.) Calls to the mathematical routines are by CALLSEG or CALLREL C180 instructions, and return is via the C180 RETURN instruction.

Upon normal return, result values are returned in registers XE and XF. 64-bit results (type INTEGER and SINGLE) are returned in XF. 128-bit results (type DOUBLE and COMPLEX) are returned in XE^XF (also denoted

(XE,XF)). For type DOUBLE, the most significant part will be in XE. For type COMPLEX, the the real part will be in XE.

2.6 ERROR HANDLING

Error recovery is the response of the C180 Math Library to the detection of an argument list or result outside the domain of the function. There are two modes of error recovery, depending on whether the calling sequence was call-by-reference or call-by-value.

2.6.1 CALL-BY-REFERENCE

Under call-by-reference, the Math Library will generate the special software condition MATH_LIBRARY_ERROR.

When an error occurs in a CMML function under call-by-reference, the following events occur:

1. An appropriate abnormal status is set into global variable MLV$STATUS (of type OST$STATUS).

2. The appropriate default error value (indicated in the function descriptions) is placed in the result register(s) (XF or XE^XF). Register A4 will contain the pointer to the the parameter list passed to the call-by-reference routine. Register XD will contain the number of parameters for the call-by-reference routine, for example, 1 for MLP$RSIN, 2 for MLP$RZTOZ. The User Condition Register will be cleared of all arithmetic errors.

3. Ungated routine MLP$ERROR_PROCESSOR is called with all registers saved in the save area.

4. MLP$ERROR_PROCESSOR calls PMP$CAUSE_CONDITION with user condition MATH_LIBRARY_ERROR and a pointer to the previous save area (the registers saved by the call-by-reference routine) as the condition descriptor.

5. Upon return from PMP$CAUSE_CONDITION, MLP$ERROR_PROCESSOR is exited if the returned status is normal. Otherwise PMP$ABORT is called with one of two statuses. Status MLV$STATUS is used if there is no established condition handler for MATH_LIBRARY_ERROR. Otherwise the status returned from PMP$CAUSE_CONDITION is used.

6. The call-by-reference routine immediately returns if it is returned to.

The mathematical library error numbers and message templates are listed in Appendix B. All error numbers starting with 67 which are currently

undefined are reserved for future expansion of the Math Library.

2.6.2 CALL-BY-VALUE

Under call-by-value, a trap interrupt will be generated in the attempt to evaluate the function with a bad argument list. No further support will be supplied. Note that the call-by-value linkage is designed for maximum speed when the argument list is within the domain of the function.

The error information regarding error number and error result is applicable only to the call by reference entry point. The value in the XF (or XE^XF) register is undefined in the case of a trap interrupt occurring during execution of call-by-value.

2.7 RELIABILITY AND PERFORMANCE

It is desirable that computed results be accurate to the full number of bits available to the result. Certain argument reductions may make this prohibitively expensive, e.g., that for DSIN, DCOS, DTAN where the argument exceeds 2**47. Double precision argument reduction is done in some cases for single precision functions in order to preserve precision and previous library capabilities but can influence performance.

In questions of timing versus memory requirements, differential proportional decreases in average execution time will be considered at least twice as important as the same differential proportional decreases in memory size. The disappearance of floating-point instructions which round requires extra work at certain points of algorithms. Lack of rounding in the floating-point operations makes exact duplication of results obtained with the C170 Math Library impossible, in general. As a result, programs calling math routines which are ill-conditioned with respect to use of those routines will show differences in output. In other programs, any differences will be minor.

2.8 MATHEMATICAL FUNCTION SPECIFICATIONS

In the following table, the set {N} represents the union of the sets {all standard numbers}, {0}, {Z1}, {Z2}, {Z3}. (N alone will denote the list of all members of {N}. This is done to simplify the notation for union. For example, {N,x} will denote the union of {N} and {x}.)

The set {I} is the set of all representable integers. (Again, I alone

will denote the list of all representable integers.) When the result is defined as a single or double precision number, the set {I} is the set of all single or double numbers {N} such that the decimal representation has only zeros to the right of the decimal point. The symbol " <- " is used to indicate "is a member of".

All references to "log" are natural logarithms (base e), unless otherwise indicated.

2.8.1 ABS

Function: ABS

Description: Absolute value of a single precision number.

Entry points: call-by-reference MLP$RABS, ABS
call-by-value MLP$VABS

Arguments: S1 - a single precision number.

Domain: S1 <- {all single numbers}

+

Result: R - a single precision number.

Range: R <- {all non-negative single numbers}

+

Error results: no errors are generated by ABS.

2.8.2 ACOS

Function: ACOS

Description: Inverse circular cosine of a single precision number.

Entry points:

call-by-reference		MLP$RACOS, ACOS
call-by-value		MLP$VACOS

Arguments: S1 - a single precision number.

Domain: S1 <- {n : |n| ≤ 1.}

Result: R - a single precision number.

Range: R <- {n : 0 ≤ n ≤ pi}

Error results:

Error Number	Arguments	Result
670001	S1 = +/-INDEF	+IND
670002	S1 = +/-INF	+IND
670003	|S1| > 1.	+IND

2.8.3 AIMAG

Function:	AIMAG	
Description:	Imaginary part of a complex number.	
Entry points:	call-by-reference	MLP$RAIMAG, AIMAG
	call-by-value	MLP$VAIMAG
Arguments:	Z1 - a complex number.	
Domain:	Z1 <- {all complex numbers}	
Result:	R - a single precision number.	
Range:	R <- {all single numbers}	
Error results:	no errors are generated by AIMAG	

2.8.4 AINT

Function: AINT

Description: Integer part of a single precision number. (Truncation)

Entry points:

call-by-reference	MLP$RAINT, AINT
call-by-value	MLP$VAINT

Arguments: S1 - a single precision number.

Domain: S1 <- {N}

Result: R - a single precision number.

Range: R <- {I}

Error results:

Error Number	Arguments	Result
670004	S1 = +/-INDEF	+IND
670005	S1 = +/-INF	+IND

2.8.5 ALOG

Function: ALOG

Description: Natural logarithm of a single precision number.

Entry points: call-by-reference MLP$RALOG, ALOG
call-by-value MLP$VALOG

Arguments: S1 - a single precision number.

Domain: S1 <- {n : n > 0.}

Result: R - a single precision number.

Range: R <- {n : |n| < 4095*log(2)}

Error results:

Error Number	Arguments	Result
670006	S1 = +/-INDEF	+IND
670007	S1 = +/-INF	+IND
670008	S1 = 0.	+IND
670009	S1 < 0.	+IND

2.8.6 ALOG10

Function: ALOG10

Description: Common logarithm of a single precision number.

Entry points:
- call-by-reference MLP$RALOG10, ALOG10
- call-by-value MLP$VALOG10

Arguments: S1 - a single precision number.

Domain: S1 <- {n : n > 0.}

Result: R - a single precision number.

Range: R <- {n : |n| < 4095*log(2)}

Error results:

Error Number	Arguments	Result
670010	S1 = +/-INDEF	+IND
670011	S1 = +/-INF	+IND
670012	S1 = 0.	+IND
670013	S1 < 0.	+IND

2.8.7 AMOD

Function: AMOD

Description: Remainder of a single precision quotient.

Entry points:
call-by-reference MLP$RAMOD, AMOD
call-by-value MLP$VAMOD

Arguments:
S1 - a single precision number.
S2 - a single precision number.

Domain: S1 <- {N}

and S2 <- {n : n =/ 0.}

and S1/S2 <- {N}

Result: R - a single precision number.

Range: R <- {N}

Error results:

Error Number	Arguments	Result
670014	S1 = +/-INDEF	+IND
670015	S2 = +/-INDEF	+IND
670016	S1 = +/-INF	+IND
670017	S2 = +/-INF	+IND
670018	S2 = 0.	+IND

670019	S1/S2 = +/-INF	+IND

2.8.8 ANINT

Function: ANINT

Description: Nearest integer to a single precision number.

Entry points: call-by-reference MLP$RANINT, ANINT
call-by-value MLP$VANINT

Arguments: S1 - a single precision number.

Domain: S1 <- {N}

Result: R - a single precision number.

Range: R <- {I}

Error results:

Error Number	Arguments	Result
670020	S1 = +/-INDEF	+IND
670021	S1 = +/-INF	+IND

2.8.9 ASIN

Function:	ASIN
Description:	Inverse circular sine of a single precision number.
Entry points:	call-by-reference MLP$RASIN, ASIN call-by-value MLP$VASIN
Arguments:	S1 - a single precision number.
Domain:	S1 <- {n : |n| ≤ 1.}
Result:	R - a single precision number.
Range:	R <- {n : |n| ≤ pi/2}

Error results:

Error Number	Arguments	Result
670022	S1 = +/-INDEF	+IND
670023	S1 = +/-INF	+IND
670024	|S1| > 1.	+IND

2.8.10 ATAN

Function: ATAN

Description: Inverse circular tangent of a single precision number.

Entry points:

call-by-reference	MLP$RATAN, ATAN
call-by-value	MLP$VATAN

Arguments: S1 - a single precision number.

Domain: S1 <- {N, +/-INF}

Result: R - a single precision number.

Range: R <- {n : |n| ≤ pi/2}

Error results:

Error Number	Arguments	Result
670025	S1 = +/-INDEF	+IND

2.8.11 ATAN2

Function: ATAN2

Description: Inverse circular tangent of a single precision quotient.

Entry points: call-by-reference MLP$RATAN2, ATAN2
call-by-value MLP$VATAN2

Arguments: S1 - a single precision number.
S2 - a single precision number.

Domain: S1 <- {N, +/-INF}
+
and S2 <- {N, +/-INF}
+
and (S1,S2) =/ (0.,0.)
+
and (S1,S2) =/ {+/INF,+/INF}
+

Result: R - a single precision number.

Range: R <- {n : -pi < n < pi}

Error results:

Error Number	Arguments	Result
670026	S1 = +/-INDEF	+IND
670027	S2 = +/-INDEF	+IND
670028	S1 = +/-INF and S2 = +/-INF	+IND
670029	S1 = S2 = 0.	+IND
670030	S1/S2 = +/-INF and S2 =/ 0	+IND

2.8.12 ATANH

Function: ATANH

Description: Inverse hyperbolic tangent of a single precision number.

Entry points: call-by-reference MLP$RATANH, ATANH
call-by-value MLP$VATANH

Arguments: S1 - a single precision number.

Domain: S1 <- {n : |n| < 1.}

Result: R - a single precision number.

Range: R <- {N}

Error results:

Error Number	Arguments	Result
670031	S1 = +/-INDEF	+IND
670032	S1 = +/-INF	+IND
670033	¦S1¦ > 1.	+IND

2.8.13 CABS

Function: CABS

Description: Absolute value of a complex number.

Entry points:

call-by-reference	MLP$RCABS, CABS
call-by-value	MLP$VCABS

Arguments: Z1 - a complex number.

Domain: Z1 <- {(n1,n2) : (n1**2 + n2**2)**1/2 <- {N}}

Result: R - a single precision number.

Range: R <- {N}

Error results:

Error Number	Arguments	Result
670034	Z1 = +/-INDEF	(+IND, +IND)
670035	Z1 = +/-INF	(+IND, +IND)
670036	|Z1| = +INF	(+IND, +IND)

2.8.14 CCOS

Function: CCOS

Description: Circular cosine of a complex number.

Entry points: call-by-reference MLP$RCCOS, CCOS
call-by-value MLP$VCCOS

Arguments: Z1 - a complex number.

Domain: Re(Z1) <- {n : |n| < 2**47}

Im(Z1) <- {n : |n| < 4095*log(2)}

Result: R - a complex number.

Range: R <- {(N,N)}

Error results:

Error Number	Arguments	Result
670037	Z1 = +/-INDEF	(+IND, +IND)
670038	Z1 = +/-INF	(+IND, +IND)
670039	|Re(Z1)| > 2**47	(+IND, +IND)
670040	Im(Z1) $\geq$ 4095*log(2)	(+IND, +IND)
670041	Im(Z1) $\leq$ -4095*log(2)	(0., 0.)

2.8.15 CEXP

Function: CEXP

Description: Exponential function of a complex number.

Entry points: call-by-reference MLP$RCEXP, CEXP

	call-by-value	MLP$VCEXP

Arguments: Z1 - a complex number.

Domain: Im(Z1) <- {n : |n| < 2**47}

Re(Z1) <- {n : n < 4095*log(2) and n > -4095*log(2)}

Result: R - a complex number.

Range: R <- {(N,N)}

Error results:

Error Number	Arguments	Result
670042	Z1 = +/-INDEF	(+IND, +IND)
670043	Z1 = +/-INF	(+IND, +IND)
670044	|Im(Z1)| ≥ 2**47	(+IND, +IND)
670045	|Re(Z1)| ≥ 4095*log(2)	(+IND, +IND)

2.8.16 CLOG

Function: CLOG

Description: Natural logarithm of a complex number.

Entry points: call-by-reference MLP$RCLOG, CLOG
call-by-value MLP$VCLOG

Arguments: Z1 - a complex number.

Domain: Z1 <- {(n1,n2) : (n1**2 + n2**2)**1/2 <- {N}}

Z1 <- {(n1,n1) : (n1,n2) =/ (0.,0.)}

Result: R - a complex number.

Range: Re(R) <- {N}

Im(R) <- {n : -pi < n < pi}

Error results:

Error Number	Arguments	Result
670046	Z1 = +/-INDEF	(+IND, +IND)
670047	Z1 = +/-INF	(+IND, +IND)
670048	|Z1| = +INF	(+IND, +IND)
670049	Z1 = (0.,0.)	(+IND, +IND)

2.8.17 CONJG

Function: CONJG

Description: Conjugate of a complex number.

Entry points: call-by-reference MLP$RCONJG, CONJG
call-by-value MLP$VCONJG

Arguments: Z1 - a complex number.

Domain: Z1 -< {all complex numbers}

+

Result: R - a complex number.

Range: R <- {all complex numbers}

+

Error results: no errors are generated by CONJG.

2.8.18 COS

Function: COS

Description: Circular cosine of a single precision number.

Entry points:	call-by-reference	MLP$RCOS, COS
	call-by-value	MLP$VCOS

Arguments: S1 - a single precision number.

Domain: S1 <- {n : |n| < 2**47}

Result: R - a single precision number.

Range: R <- {n : |n| ≤ 1.}

Error results:

Error Number	Arguments	Result
670050	S1 = +/-INDEF	+IND
670051	S1 = +/-INF	+IND
670052	S1 ≥ 2**47	+IND

2.8.19 COSD

Function: COSD

Description: Circular cosine of a single precision number in degrees.

Entry points: call-by-reference MLP$RCOSD, COSD
call-by-value MLP$VCOSD

Arguments: S1 - a single precision number

Domain: S1 <- {n : |n| < 2**47}

+

Result: R - a single precision number

Range: R <- {n : |n| < 1.}
+ -

Error results:

Error Number	Arguments	Result
670247	S1 = +/-INDEF	+IND
670248	S1 = +/-INF	+IND
670249	|S1| > 2**47	+IND

+ -

2.8.20 COSH

Function: COSH

Description: Hyperbolic cosine of a single precision number.

Entry points:

	call-by-reference	MLP$RCOSH, COSH
	call-by-value	MLP$VCOSH

Arguments: S1 - a single precision number.

Domain: S1 <- {n : |n| < 4095*log(2)}

Result: R - a single precision number.

Range: R <- {N}

Error results:

Error Number	Arguments	Result
670053	S1 = +/-INDEF	+IND
670054	S1 = +/-INF	+IND
670055	|S1| > 4095*log(2)	+IND

2.8.21 COTAN

Function: COTAN

Description: Circular cotangent of a single precision number.

Entry points:

	call-by-reference	MLP$RCOTAN
	call-by-value	MLP$VCOTAN

Arguments: S1 - a single precision number.

Domain: S1 <- {n : 0. < |n| < 2**47}

Result: R - a single precision number.

Range: R <- {N}

Error results:

Error number	Arguments	Result
670254	S1 = +/-INDEF	+IND
670255	S1 = +/-INF	+IND
670256	S1 >= 2**47	+IND
670265	S1 = 0.	+IND

2.8.22 CSIN

Function: CSIN

Description: Circular sine of a complex number.

Entry points: call-by-reference MLP$RCSIN, CSIN
call-by-value MLP$VCSIN

Arguments: Z1 - a complex number.

Domain: Re(Z1) <- {n : ¦n¦ < 2**47}

Im(Z1) <- {n : ¦n¦ < 4095*log(2)}

Result: R - a complex number.

Range: R <- {(N,N)}

Error results:

Error Number	Arguments	Result
670056	Z1 = +/-INDEF	(+IND, +IND)
670057	Z1 = +/-INF	(+IND, +IND)
670058	¦Re(Z1)¦ > 2**47	(+IND, +IND)
670059	¦Im(Z1)¦ ≥ 4095*log(2)	(+IND, +IND)

2.8.23 CSQRT

Function: CSQRT

Description: Square root of a complex number.

Entry points: call-by-reference MLP$RCSQRT, CSQRT
call-by-value MLP$VCSQRT

Arguments: Z1 - a complex number.

Domain Z1 <- {(n1,n2) : ((n1**2 + n2**2)**1/2) + ¦n1¦ <- {N}}
+

Result: R - a complex number.

Range: R <- {(n1,n2) : n1 ≥ 0.}
+

Error results:

Error Number	Arguments	Result
670060	Z1 = +/-INDEF	(+IND, +IND)
670061	Z1 = +/-INF	(+IND, +IND)
670062	¦Z1¦+¦n1¦ = +INF	(+IND, +IND)

2.8.24 DABS

Function: DABS

Description: Absolute value of a double precision number.

Entry points: call-by-reference MLP$RDABS, DABS
call-by-value MLP$VDABS

Arguments: D1 - a double precision number.

Domain: D1 <- {all double numbers}

+

Result: R - a double precision number.

Range: R <- {all non-negative double-precision numbers}

+

Error results: no errors are generated by DABS

2.8.25 DACOS

Function: DACOS

Description: Inverse circular cosine of a double precision number.

Entry points:

	call-by-reference	MLP$RDACOS, DACOS
	call-by-value	MLP$VDACOS

Arguments: D1 - a double precision number.

Domain $D1 \leftarrow \{n : |n| \le 1.\}$

Result: R - a double precision number.

Range: $R \leftarrow \{n : 0 \le n \le pi\}$

Error results:

Error Number	Arguments	Result
670063	D1 = +/-INDEF	(+IND, +IND)
670064	D1 = +/-INF	(+IND, +IND)
670065	|D1| > 1.	(+IND, +IND)

2.8.26 DASIN

Function: DASIN

Description: Inverse circular sine of a double precision number.

Entry points: call-by-reference MLP$RDASIN, DASIN
call-by-value MLP$VDASIN

Arguments: D1 - a double precision number.

Domain: D1 <- {n : |n| ≤ 1.}

Result: R - a double precision number.

Range: R <- {n : |n| ≤ pi/2}

Error results:

Error Number	Arguments	Result
670066	D1 = +/-INDEF	(+IND, +IND)
670067	D1 = +/-INF	(+IND, +IND)
670068	|D1| > 1.	(+IND, +IND)

2.8.27 DATAN

Function: DATAN

Description: Inverse circular tangent of a double precision number.

Entry points: call-by-reference MLP$RDATAN, DATAN
call-by-value MLP$VDATAN

Arguments: D1 - a double precision number.

Domain: D1 <- {N, +/-INF}

+

Result: R - a double precision number.

Range: R <- {n : |n| ≤ pi/2}

+

Error results:

+

Error Number	Arguments	Result
670069	D1 = +/-INDEF	+IND

2.8.28 DATAN2

Function: DATAN2

Description: Inverse circular tangent of a double precision quotient.

Entry points: call-by-reference MLP$RDATAN2, DATAN2
call-by-value MLP$VDATAN2

Arguments: D1 - a double precision number.
D2 - a double precision number.

Domain: D1 <- {N, +/-INF}

and D2 <- {N, +/-INF}

and (D1,D2) =/ (0.,0.)

Result: R - a double precision number.

Range: R <- {n : -pi < n < pi}

Error results:

Error Number	Arguments	Result
670070	D1 = +/-INDEF	(+IND, +IND)
670071	D2 = +/-INDEF	(+IND, +IND)
670072	D1 = D2 = +/-INF	(+IND, +IND)
670073	D1 = D2 = 0.	(+IND, +IND)

2.8.29 DCOS

Function: DCOS

Description: Circular cosine of a double precision number.

Entry points: call-by-reference MLP$RDCOS, DCOS
call-by-value MLP$VDCOS

Arguments: D1 - a double precision number.

Domain: D1 <- {n : !n! < 2**47}

Result: R - a double precision number.

Range: R <- {n : !n! < 1.}

Error results:

Error Number	Arguments	Result
670074	D1 = +/-INDEF	(+IND, +IND)
670075	D1 = +/-INF	(+IND, +IND)
670076	!D1! > 2**47	(+IND, +IND)

2.8.30 DCOSH

Function: DCOSH

Description: Hyperbolic cosine of a double precision number.

entry points: call-by-reference MLP$RDCOSH, DCOSH
call-by-value MLP$VDCOSH

Arguments: D1 - a double precision number.

Domain: D1 <- {n : |n| < 4095*log(2)}

+

Result: R - a double precision number.

Range: R <- {N}

+

Error results:

+

Error Number	Arguments	Result
670077	D1 = +/-INDEF	(+IND, +IND)
670078	D1 = +/-INF	(+IND, +IND)
670079	|D1| > 4095*log(2)	(+IND, +IND)

2.8.31 DDIM

Function:	DDIM	
Description:	Positive difference of two double precision numbers.	
Entry points:	call-by-reference	MLP$RDDIM, DDIM
	call-by-value	MLP$VDDIM
Arguments:	D1 - a double precision number.	
	D2 - a double precision number.	
Domain:	D1 <- {N}	
and	D2 <- {N}	
and	D1 - D2 <- {N}	
Result:	R - a double precision number.	
Range:	R <- {n : n > 0.}	

Error results:

Error Number	Arguments	Result
670080	D1 = +/-INDEF	(+IND, +IND)
670081	D2 = +/-INDEF	(+IND, +IND)
670082	D1 = +/-INF	(+IND, +IND)
670083	D2 = +/-INF	(+IND, +IND)
670084	D1 - D2 = +/-INF	(+IND, +IND)

2.8.32 DEXP

Function: DEXP

Description: Exponential function of a double precision number.

Entry points: call-by-reference MLP$RDEXP, DEXP
call-by-value MLP$VDEXP

Arguments: D1 - a double precision number.

Domain: D1 <- {n : |n| < 4095*log(2)}

Result: R - a double precision number.

Range: R <- {N}

Error results:

Error Number	Arguments	Result
670085	D1 = +/-INDEF	(+IND, +IND)
670086	D1 = +/-INF	(+IND, +IND)
670087	|D1| ≥ 4095*log(2)	(+IND, +IND)
670088	|D1| ≤ -4095*log(2)	(0., 0.)

2.8.33 DIM

Function: DIM

Description: Positive difference of two single precision numbers.

Entry points: call-by-reference MLP$RDIM, DIM
call-by-value MLP$VDIM

Arguments: S1 - a single precision number.
S2 - a single precision number.

Domain: S1 <- {N}

and S2 <- {N}

and S1 - S2 <- {N}

Result: R - a single precision number.

Range: R <- {N}

Error results:

Error Number	Arguments	Result
670089	S1 = +/-INDEF	+IND
670090	S2 = +/-INDEF	+IND
670091	S1 = +/-INF	+IND
670092	S2 = +/-INF	+IND
670093	S1 - S2 = +/-INF	+IND

2.8.34 DINT

Function: DINT

Description: Integer (whole number) part of a double precision number.
(Truncation.)

Entry points: call-by-reference MLP$RDINT, DINT
call-by-value MLP$VDINT

Arguments: D1 - a double precision number.

Domain: D1 <- {N}

Result: R - a double precision number.

Range: R <- {I}

Error results:

Error Number	Arguments	Result
670094	D1 = +/-INDEF	(+IND, +IND)
670095	D1 = +/-INF	(+IND, +IND)

2.8.35 DLOG

Function: DLOG

Description: Natural logarithm of a double precision number.

Entry points:

	call-by-reference	MLP$RDLOG, DLOG
	call-by-value	MLP$VDLOG

Arguments: D1 - a double precision number.

Domain: D1 <- {n : n > 0.}

Result: R - a double precision number.

Range: R <- {n : |n| < 4095*log(2)}

Error results:

Error Number	Arguments	Result
670096	D1 = +/-INDEF	(+IND, +IND)
670097	D1 = +/-INF	(+IND, +IND)
670098	D1 = 0.	(+IND, +IND)
670099	D1 < 0.	(+IND, +IND)

2.8.36 DLOG10

Function: DLOG10

Description: Common logarithm of a double precision number.

Entry points: call-by-reference   MLP$RDLOG10, DLOG10
                 call-by-value   MLP$VDLOG10

Arguments: D1 - a double precision number.

Domain: D1 <- {n : n > 0.}

Result: R - a double precision number.

Range: R <- {n : |n| < 4095*log(2)}

Error results:

Error Number	Arguments	Result
670100	D1 = +/-INDEF	(+IND, +IND)
670101	D1 = +/-INF	(+IND, +IND)
670102	D1 = 0.	(+IND, +IND)
670103	D1 < 0.	(+IND, +IND)

2.8.37 DMOD

Function: DMOD

Description: Remainder of a double precision quotient.

Entry points:	call-by-reference	MLP$RDMOD, DMOD
	call-by-value	MLP$VDMOD

Arguments: D1 - a double precision number.
D2 - a double precision number.

Domain: D1 <- {N}

and D2 <- {n : n =/ 0.}

and D1 / D2 <- {N}

Result: R - a double precision number.

Range: R <- {N}

Error results:

Error Number	Arguments	Result
670104	D1 = +/-INDEF	(+IND, +IND)
670105	D2 = +/-INDEF	(+IND, +IND)
670106	D1 = +/-INF	(+IND, +IND)
670107	D2 = +/-INF	(+IND, +IND)
670108	D2 = 0.	(+IND, +IND)
670109	D1 / D2 = +/-INF	(+IND, +IND)

2.8.38 DNINT

Function: DNINT

Description: Nearest whole number to a double precision number.

Entry points:	call-by-reference	MLP$RDNINT, DNINT
	call-by-value	MLP$VDNINT

Arguments: D1 - a double precision number.

Domain: D1 <- {N}

+

Result: R - a double precision number.

Range: R <- {I}

+

Error results:

+

Error Number	Arguments	Result
670110	D1 = +/-INDEF	(+IND, +IND)
670111	D1 = +/-INF	(+IND, +IND)

2.8.39 DPROD

Function:	DPROD	
Description:	Product of two double precision numbers.	
Entry points:	call-by-reference	MLP$RDPROD, DPROD
	call-by-value	MLP$VDPROD
Arguments:	D1 - a double precision number.	
	D2 - a double precision number.	
Domain:	D1 <- {N}	
and	D2 <- {N}	
and	D1*D2 <- {N}	
Result:	R - a double precision number.	
Range:	R <- {N}	

Error results:

Error Number	Arguments	Result
670112	D1 = +/-INDEF	(+IND, +IND)
670113	D2 = +/-INDEF	(+IND, +IND)
670114	D1 = +/-INF	(+IND, +IND)
670115	D2 = +/-INF	(+IND, +IND)
670116	D1 * D2 = +/-INF	(+IND, +IND)

2.8.40 DSIGN

Function: DSIGN

Description: Double precision transfer of sign.

Entry points: call-by-reference MLP$RDSIGN, DSIGN
call-by-value MLP$VDSIGN

Arguments: D1 - a double precision number.
D2 - a double precision number.

Domain: D1 <- {all double numbers}
+
and D2 <- {all double numbers}
+

Result: R - a double precision number.

Range: R <- {all double numbers}
+

Error results: no errors are generated by DSIGN

2.8.41 DSIN

Function: DSIN

Description: Circular sine of a double precision number.

Entry points: call-by-reference MLP$RDSIN, DSIN
call-by-value MLP$VDSIN

Arguments: D1 - a double precision number.

Domain: D1 <- {n : |n| < 2**47}

+

Result: R - a double precision number.

Range: R <- {n : |n| ≤ 1.}

+

Error results:

+

Error Number	Arguments	Result
670117	D1 = +/-INDEF	(+IND, +IND)
670118	D1 = +/-INF	(+IND, +IND)
670119	|D1| ≥ 2**47	(+IND, +IND)

+

2.8.42 DSINH

Function: DSINH

Description: Hyperbolic sine of a double precision number.

Entry points:

	call-by-reference	MLP$RDSINH, DSINH
	call-by-value	MLP$VDSINH

Arguments: D1 - a double precision number.

Domain: D1 <- {n : ¦n¦ < 4095*log(2)}

Result: R - a double precision number.

Range: R <- {N}

Error results:

Error Number	Arguments	Result
670120	D1 = +/-INDEF	(+IND, +IND)
670121	D1 = +/-INF	(+IND, +IND)
670122	¦D1¦ > 4095*log(2)	(+IND, +IND)

2.8.43 DSQRT

Function:	DSQRT	
Description:	Square root of a double precision number.	
Entry points:	call-by-reference	MLP$RDSQRT, DSQRT
	call-by-value	MLP$VDSQRT
Arguments:	D1 - a double precision number.	
Domain:	D1 <- {n : n ≥ 0.}	
Result:	R - a double precision number.	
Range:	R <- {N}	

Error results:

Error Number	Arguments	Result
670123	D1 = +/-INDEF	(+IND, +IND)
670124	D1 = +/-INF	(+IND, +IND)
670125	D1 < 0.	(+IND, +IND)

2.8.44 DTAN

Function: DTAN

Description: Circular tangent of a double precision number.

Entry points: call-by-reference MLP$RDTAN, DTAN
call-by-value MLP$VDTAN

Arguments: D1 - a double precision number.

Domain: D <- {n : !n! < 2**47}

Result: R - a double precision number.

Range: R <- {N}

Error results:

Error Number	Arguments	Result
670126	D1 = +/-INDEF	(+IND, +IND)
670127	D1 = +/-INF	(+IND, +IND)
670128	!D1! > 2**47	(+IND, +IND)

2.8.45 DTANH

Function: DTANH

Description: Hyperbolic tangent of a double precision number.

Entry points: call-by-reference MLP$RDTANH, DTANH
call-by-value MLP$VDTANH

Arguments: D1 - a double precision number.

Domain: D1 <- {N, +/-INF}

Result: R - a double precision number.

Range: R <- {n : |n| ≤ 1.}

Error results:

Error Number	Arguments	Result
670129	D1 = +/-INDEF	(+IND, +IND)

2.8.46 DTOD

Function: DTOD

Description: Raise a double precision base to a double precision power.

Entry points:

	call-by-reference	MLP$RDTOD, DTOD
	call-by-value	MLP$VDTOD

Arguments:

D1 - a double precision number.
D2 - a double precision number.

Domain:

D1 <- {n : n ≥ 0.}

and D2 <- {N}

and if D1 = 0, D2 > 0
and D1**D2 <- {N}

Result: R - a double precision number.

Range: R <- {N}

Error results:

Error Number	Arguments	Result
670130	D1 = +/-INDEF	(+IND, +IND)
670131	D2 = +/-INDEF	(+IND, +IND)
670132	D1 = +/-INF	(+IND, +IND)
670133	D2 = +/-INF	(+IND, +IND)
670134	D1 = 0. and D2 ≤ 0.	(+IND, +IND)
670135	D1 < 0.	(+IND, +IND)
670136	D1**D2 = +/-INF	(+IND, +IND)

2.8.47 DTOI

Function: DTOI

Description: Raise a double precision base to an integer power.

Entry points:
- call-by-reference MLP$RDTOI, DTOI
- call-by-value MLP$VDTOI

Arguments:
- D1 - a double precision number.
- I2 - an integer.

Domain: D1 <- {N}

and I2 <- {all integers}

and if D1 = 0, I2 > 0

Result: R - a double precision number.

Range: R <- {N}

Error results:

Error Number	Arguments	Result
670137	D1 = +/-INDEF	(+IND, +IND)
670138	D1 = +/-INF	(+IND, +IND)
670139	D1 = 0. and I2 < 0	(+IND, +IND)
670140	D1**I2 = +/-INF	(+IND, +IND)

2.8.48 DTOX

Function: DTOX

Description: Raise a double precision base to a single precision power.

Entry points: call-by-reference MLP$RDTOX, DTOX
call-by-value MLP$VDTOX

Arguments: D1 - a double precision number.
S2 - a single precision number.

Domain: D1 <- {n : n ≥ 0.)

+ and D2 <- {N}

+ and if D1 = 0, S2 > 0.

Result: R - a double precision number.

Range: R <- {N}

+

Error results:

+

Error Number	Arguments	Result
670141	D1 = +/-INDEF	(+IND, +IND)
670142	S2 = +/-INDEF	(+IND, +IND)
670143	D1 = +/-INF	(+IND, +IND)

670144	S2 = +/-INF	(+IND, +IND)
670145	D1 = 0 and S2 < 0.	(+IND, +IND)
670146	D1 < 0.	(+IND, +IND)
670147	D1**S2 = +/-INF	(+IND, +IND)

2.8.49 DTOZ

Function: DTOZ

Description: Raise a double precision base to a complex power.

Entry points:
call-by-reference MLP$RDTOZ, DTOZ
call-by-value MLP$VDTOZ

Arguments:
D1 - a double precision number.
Z2 - a complex number.

Domain: D1 <- {N}

and Z2 <- {(N,N)}

and if D1 = 0., Z2 <- {(n1,n2) : n1 > 0., n2 = 0.}

Result: R - a complex number.

Range: R <- {(N,N)}

Error results:

Error Number	Arguments	Result
670148	D1 = +/-INDEF	(+IND, +IND)
670149	Z2 = +/-INDEF	(+IND, +IND)
670150	D1 = +/-INF	(+IND, +IND)
670151	Z2 = +/-INF	(+IND, +IND)
670152	D1 = 0.	
and	Re(Z2) < 0. or Im(Z2) =/ 0.	(+IND, +IND)
670153	D1 ≤ 0.	(+IND, +IND)
670154	D1**Z2 = +/-INF	(+IND, +IND)

2.8.50 ERF

Function: ERF

Description: Error function of a single precision number.

Entry points: call-by-reference MLP$RERF, ERF
call-by-value MLP$VERF

Arguments: S1 - a single precision number.

Domain: S1 <- {N}

Result: R - a single precision number.

Range: R <- {n : -1. ≤ n ≤ 1.}

Error results:

Error Number	Arguments	Result

670155	S1 = +/-INDEF	+IND

2.8.51 ERFC

Function: ERFC

Description: Error function complement of a single precision number.

Entry points:

	call-by-reference	MLP$RERFC, ERFC
	call-by-value	MLP$VERFC

Arguments: S1 - a single precision number.

Domain: S1 <- {n : n < 25.923}

+

Result: R - a single precision number.

Range: R <- {n : 0. < n < 2.}
- -

+

Error results:

Error Number	Arguments	Result
670156	S1 = +/-INDEF	+IND
670184	S1 ≥ 25.923	0.

2.8.52 EXP

Function: EXP

Description: Exponential function of a single precision number.

Entry points: call-by-reference MLP$REXP, EXP
call-by-value MLP$VEXP

Arguments: S1 - a single precision number.

Domain: S1 <- {n : |n| < 4095*log(2)}

Result: R - a single precision number.

Range: R <- {N}

Error results:

Error Number	Arguments	Result
670157	S1 = +/-INDEF	+IND
670158	S1 = +/-INF	+IND
670159	S1 > 4095*log(2)	+IND
670160	S1 $\leq$ -4095*log(2)	0.

2.8.53 EXTB

Function: EXTB

Description: EXTB(a,i1,i2) - Extracts bits from argument a, as indicated by i1 and i2. Argument i1 indicates the first bit to be extracted, numbering from bit zero on the left. Argument i2 indicates the number of bits to be extracted.

Entry points:

call-by-reference	MLP$REXTB
call-by-value	MLP$VEXTB

Arguments:	The parameter,a is any data type except character or bit. For a double precision or complex argument a, the argument used is REAL(A). i1 and i2 are integers.
Domain:	i1,i2 <- {i1,i2: i1 + i2 <= 64} a <- {REALS} OR a <- {DOUBLE PRECISION NUMBERS} OR a <- {INTEGERS} OR a <- {COMPLEX NUMBERS}
Result:	R - a FORTRAN type BOOLEAN value (64-bit word).
Range:	R <- {BOOLEAN}

Error results:

Error number	Arguments	Result
670257	i1 < 0	+IND
670258	i2 < 0	+IND
670259	i1 >= 64	+IND
670260	i1 + i2 > 64	+IND

2.8.54 IABS

Function:	IABS	
Description:	Absolute value of an integer.	
Entry points:	call-by-reference	MLP$RIABS, IABS
	call-by-value	MLP$VIABS
Arguments:	I1 - an integer.	
Domain:	I1 <- {all integers}	
Result:	R - an integer.	

Range: R <- {i : i > 0}
+ -

Error results: no errors are generated by IABS

2.8.55 IDIM

Function: IDIM

Description: Positive difference of two integers.

Entry points: call-by-reference MLP$RIDIM, IDIM
call-by-value MLP$VIDIM

Arguments: I1 - an integer.
I2 - an integer.

Domain: (I1, I2) <- {(i1, i2) : i1 - i2 < 2**63}
+

Result: R - an integer.

Range: R <- {i : i ≥ 0}

Error results:

Error Number	Arguments	Result
670161	I1 - I2 ≥ 2**63	0

2.8.56 IDNINT

Function: IDNINT

Description: Nearest whole number to a double precision number.

Entry points: call-by-reference MLP$RIDNINT, IDNINT
call-by-value MLP$VIDNINT

Arguments: D1 - a double precision number.

Domain: D1 <- {N}

Result: R - an integer.

Range: R <- {I}

Error results:

Error Number	Arguments	Result
670162	D1 = +/-INDEF	0
670163	D1 = +/-INF	0

2.8.57 INSB

Function: INSB

Description: INSB(a,i1,i2,b) - Inserts bits from argument a (rightmost i2 bits) into copy of b (beginning with bit position i1, length = i2 bits).

Entry points: call-by-reference MLP$RINSB
call-by-value MLP$VINSB

Arguments: The parameters a,b are any data type except character or bit. For double precision or complex arguments a,b; the arguments used are REAL(a) and REAL(b) respectively. i1 and i2 are integers.

Domain: i1,i2 <- {i1,i2: i1 + i2 <= 64}
a,b <- {REALS} OR a,b <- {DOUBLE PRECISION NUMBERS} OR
a,b <- {INTEGERS} OR a,b <- {COMPLEX NUMBERS}

Result: R - a FORTRAN type BOOLEAN value (64-bit word).

Range: R <- {BOOLEAN}

Error results:

Error number	Arguments	Result
670261	i1 < 0	+IND
670262	i2 < 0	+IND
670263	i1 >= 64	+IND
670264	i1 + i2 > 64	+IND

2.8.58 ISIGN

Function: ISIGN

Description: Integer transfer of sign.

Entry points: call-by-reference MLP$RISIGN, ISIGN
call-by-value MLP$VISIGN

Arguments: I1 - an integer.

	I2 - an integer.
Domain:	I1 <- {all integers}
and	I2 <- {all integers}
Result:	R - an integer.
Range:	R <- {all integers}
Error results:	no errors are generated by ISIGN

2.8.59 ITOD

Function:	ITOD	
Description:	Raise an integer base to a double precision power.	
Entry points:	call-by-reference	MLP$RITOD, ITOD
	call-by-value	MLP$VITOD

Arguments:	I1 - an integer.
	D2 - a double precision number.
Domain:	I1 <- {i : i $\geq$ 0}
and	D2 <- {N}
and	if I1 = 0, D2 > 0.
Result:	R - a double precision number.
Range:	R <- {N}

Error results:

Error Number	Arguments	Result
670164	D2 = +/-INDEF	(+IND, +IND)
670165	D2 = +/-INF	(+IND, +IND)
670166	I1 = 0 and D2 $\leq$ 0.	(+IND, +IND)
670167	I1 < 0	(+IND, +IND)
670168	I1**D2 = +/-INF	(+IND, +IND)

2.8.60 ITOI

Function: ITOI

Description: Raise an integer base to an integer power.

Entry points: call-by-reference MLP$RITOI, ITOI
call-by-value MLP$VITOI

Arguments: I1 - an integer.
I2 - an integer.

Domain: I1 <- {all integers}

and I2 <- {all integers}

and if I1 = 0, I2 > 0
and |I1**I2| < 2**63

Result: R - an integer.

Range: R <- {all integers}

Error results:

Error Number	Arguments	Result
670169	|I1**I2| > 2**63	0
670170	I1 = 0 and I2 < 0	0

2.8.61 ITOX

Function:	ITOX
Description:	Raise an integer base to a single precision power.
Entry points:	call-by-reference MLP$RITOX, ITOX call-by-value MLP$VITOX
Arguments:	I1 - an integer. S2 - a single precision number.
Domain:	I1 <- {i : i $\geq$ 0}
and	S2 <- {N}
and	if I1 = 0, S2 > 0.
Result:	R - a single precision number.
Range:	R <- {N}

Error results:

Error Number	Arguments	Result
670171	S2 = +/-INDEF	+IND
670172	S2 = +/-INF	+IND
670173	I1 = 0 and S2 $\leq$ 0.	+IND
670174	I1 < 0	+IND
670175	I1**S2 = +/-INF	+IND

2.8.62 ITOZ

Function: ITOZ

Description: Raise an integer base to a complex power.

Entry points: call-by-reference MLP$RITOZ, ITOZ
call-by-value MLP$VITOZ

Arguments: I1 - an integer.
Z2 - a complex number.

Domain: I1 <- {n : n > 0}

and Z2 <- {(N,N)}

and if I1 = 0, Z2 <- {(n1,n2) : n1 > 0., n2 = 0.}

Result: R - a complex number.

Range: R <- {(N,N)}

Error results:

Error Number	Arguments	Result
670176	Z2 = +/-INDEF	(+IND, +IND)
670177	Z2 = +/-INF	(+IND, +IND)
670178	I1 = 0	
and	Re(Z2) < 0. or Im(Z2) =/ 0.	(+IND, +IND)
670179	I1**Z2 = +/-INF	(+IND, +IND)
670180	I1 < 0	(+IND, +IND)

2.8.63 MOD

Function: MOD

Description: Remainder of an integer quotient.

Entry points: call-by-reference MLP$RMOD, MOD
call-by-value MLP$VMOD

Arguments: I1 - an integer.
I2 - an integer.

Domain: I1 <- {all integers}

and I2 <- {i : i =/ 0}

Result: R - an integer.

Range: R <- {all integers}

Error results:

Error Number	arguments	Result
670181	I2 = 0	0

2.8.64 NINT

Function: NINT

Description: Nearest whole number to a single precision number.

Entry points:

	call-by-reference	MLP$RNINT, NINT
	call-by-value	MLP$VNINT

Arguments: S1 - a single precision number.

Domain: S1 <- {N}

Result: R - an integer.

Range: R <- {I}

Error results:

Error Number	Arguments	Result
670182	S1 = +/-INDEF	0
670183	S1 = +/-INF	0

2.8.65 RANF

Function: RANF

Description: Random number generator (single precision).

Entry points:

call-by-reference	MLP$RRANF, RANF
call-by-value	MLP$VRANF

Arguments: there is no argument to RANF.

Domain: not applicable.

Result: R - a single precision number.

Range: R <- {n : 0. < n < 1.}

Error results: no errors are generated by RANF

Comments: RANF is intended to return the same values as the RANF implemented on the 170 machines as long as the (default) initial value provided by the two libraries is used by the caller. The values of the random number seed and multiplier used in the Math Library random number generation routines, RANF, RANGET and RANSET, are made available to host languages in RANDATA, a data-only module in the Math Library. The values contained in this module are:

Value	Definition
• mlv$initial_seed	default initial seed
• mlv$random_seed	current random seed
• mlv$random_multiplier	random multiplier

The initial value of both mlv$initial_seed and mlv$random_seed is 40002BC68CFE166D(16). The initial value of mlv$random_multiplier is 40302875A2E7B175(16). The algorithm does not change the values of mlv$initial_seed or mlv$random_multiplier, and no user-callable routines are provided to change them.

2.8.66 RANGET

Procedure: RANGET

Description: Get the random number seed (a single precision number).

Entry points: call-by-reference RANGET
There is no call-by-value entry for RANGET.

Arguments: R - a single precision number
(the argument receives the result)

Domain: not applicable

Result: R - the argument.

Range: to be supplied.

Error results: no errors are generated by RANGET

2.8.67 RANSET

Routine: RANSET

Description: Set the random number seed (a single precision number).

Entry points: call-by-reference RANSET
There is no call-by-value entry for RANSET.

Arguments: S1 - a single precision number.

Domain: S1 <- {n : 0. < n < 1.}

Result: not applicable.

Range: not applicable

Error results: no errors are generated by RANSET.

2.8.68 SIGN

Function: SIGN

Description: Single precision transfer of sign.

Entry points: call-by-reference MLP$RSIGN, SIGN
call-by-value MLP$VSIGN

Arguments: S1 - a single precision number.
S2 - a single precision number.

Domain: S1 <- {all single numbers}
+
and S2 <- {all single numbers}
+

Result: R - a single precision number.

Range: R <- {n : n ≥ 0.}
+

Error results: no errors are generated by SIGN

2.8.69 SIN

Function: SIN

Description: Circular sine of a single precision number.

Entry points:

	call-by-reference	MLP$RSIN, SIN
	call-by-value	MLP$VSIN

Arguments: S1 - a single precision number.

Domain: S1 <- {n : |n| < 2**47}

Result: R - a single precision number.

Range: R <- {n : |n| ≤ 1.}

Error results:

Error Number	Arguments	Result
670185	S1 = +/-INDEF	+IND
670186	S1 = +/-INF	+IND
670187	|S1| ≥ 2**47	+IND

2.8.70 SIND

Function:	SIND	
Description:	Circular sine of a single precision number in degrees.	
Entry points:	call-by-reference	ML P$RSIND, SIND
	call-by-value	MLP$VSIND
Arguments:	S1 - a single precision number.	
Domain:	S1 <- {n : |n| < 2**47}	
Result:	R - a single precision number.	
Range:	R <- {n : |n| < 1.}	

Error results:

Error Number	Arguments	Result
670244	S1 = +/-INDEF	+IND
670245	S1 = +/-INF	+IND
670246	|S1| > 2**47	+IND

2.8.71 SINH

Function: SINH

Description: Hyperbolic sine of a single precision number.

Entry points:

call-by-reference		MLP$RSINH, SINH
call-by-value		MLP$VSINH

Arguments: S1 - a single precision number.

Domain: S1 <- {n : |n| < 4095*log(2)}

+

Result: R - a single precision number.

Range: R <- {N}

+

Error results:

Error Number	Arguments	Result
670188	S1 = +/-INDEF	+IND
670189	S1 = +/-INF	+IND
670190	|S1| > 4095*log(2)	+IND

+

2.8.72 SQRT

Function: SQRT

Description: Square root of a single precision number.

Entry points:
- call-by-reference — MLP$RSQRT, SQRT
- call-by-value — MLP$VSQRT

Arguments: S1 - a single precision number.

Domain: S1 <- {n : n ≥ 0.}

Result: R - a single precision number.

Range: R <- {n : n ≥ 0.}

Error results:

Error Number	Arguments	Result
670191	S1 = +/-INDEF	+IND
670192	S1 = +/-INF	+IND
670193	S1 < 0.	+IND

2.8.73 TAN

Function: TAN

Description: Circular tangent of a single precision number.

Entry points: call-by-reference MLP$RTAN, TAN
call-by-value MLP$VTAN

Arguments: S1 - a single precision number.

Domain: S1 <- {n : !n! < 2**47}

Result: R - a single precision number.

Range: R <- {N}

Error results:

Error Number	Arguments	Result
670194	S1 = +/-INDEF	+IND
670195	S1 = +/-INF	+IND
670196	!S1! > 2**47	+IND

2.8.74 TAND

Function: TAND

Description: Circular tangent of a single precision number in degrees.

Entry points:

call-by-reference	MLP$RTAND, TAND
call-by-value	MLP$VTAND

Arguments: S1 - a single precision number.

Domain: S1 <- {n : |n| < 2**47 and

n =/ 90*m where m <- set of odd integers}

Result: R - a single precision number.

Range: R <- {N}

Error results:

Error Number	Arguments	Result
670250	S1 = +/-INDEF	+IND
670251	S1 = +/-INF	+IND

670252	¦S1¦ > 2**47	+IND
670253	S1 is an odd multiple of 90	+IND

2.8.75 TANH

Function: TANH

Description: Hyperbolic tangent of a single precision number.

Entry points: call-by-reference MLP$RTANH, TANH
call-by-value MLP$VTANH

Arguments: S1 - a single precision number.

Domain: S1 <- {N, +/-INF}

Result: R - a single precision number.

Range: R <- {n : ¦n¦ < 1.}

Error results:

Error Number	Arguments	Result
670197	S1 = +/-INDEF	+IND

2.8.76 XTOD

Function: XTOD

Description: Raise a single precision base to a double precision power.

Entry points: call-by-reference MLP$RXTOD, XTOD
call-by-value MLP$VXTOD

Arguments: S1 - a single precision number.
D2 - a double precision number.

Domain: S1 <- {n : n ≥ 0.}

and D2 <- {N}

and if S1 = 0., D2 > 0.

Result: R - a double precision number.

Range: R <- {N}

Error results:

Error Number	Arguments	Result
670198	S1 = +/-INDEF	(+IND, +IND)
670199	D2 = +/-INDEF	(+IND, +IND)
670200	S1 = +/-INF	(+IND, +IND)
670201	D2 = +/-INF	(+IND, +IND)
670202	S1 = 0. and D2 < 0.	(+IND, +IND)
670203	S1 < 0.	(+IND, +IND)
670204	S1**D2 = +/-INF	(+IND, +IND)

2.8.77 XTOI

Function: XTOI

Description: Raise a single precision base to an integer power.

Entry points:
- call-by-reference — MLP$RXTOI, XTOI
- call-by-value — MLP$VXTOI

Arguments:
- S1 - a single precision number.
- I2 - an integer.

Domain: S1 <- {N}

and I2 <- {all integers}

and if S1 = 0, I2 > 0

Result: R - a single precision number.

Range: R <- {N}

Error results:

Error Number	Arguments	Result
670205	S1 = +/-INDEF	+IND
670206	S1 = +/-INF	+IND
670207	S1 = 0 and I2 < 0	+IND
670208	S1**I2 = +/-INF	+IND

2.8.78 XTOX

Function: XTOX

Description: Raise a single precision base to a single precision power.

Entry points:

call-by-reference		MLP$RXTOX, XTOX
call-by-value		MLP$VXTOX

Arguments: S1 - a single precision number.
S2 - a single precision number.

Domain: S1 <- {n : n > 0.}

and S2 <- {N}

and if S1 = 0., S2 > 0.
and S1**S2 <- {N}

Result: R - a single precision number.

Range: R <- {n : n ≥ 0.}

Error results:

Error Number	Arguments	Result
670209	S1 = +/-INDEF	+IND
670210	S2 = +/-INDEF	+IND
670211	S1 = +/-INF	+IND
670212	S2 = +/-INF	+IND
670213	S1 = 0. and S2 ≤ 0.	+IND
670214	S1 < 0.	+IND
670215	S1**S2 = +INF	+IND

2.8.79 XTOZ

Function: XTOZ

Description: Raise a single precision base to a complex power.

Entry points: call-by-reference MLP$RXTOZ, XTOZ
call-by-value MLP$VXTOZ

Arguments: S1 - a single precision number.

Z2 - a complex number.

Domain: S1 <- {N}

and Z2 <- {(N,N)}

and if S1 = 0, Z2 <- {(n1,n1) : n1 > 0., n2 = 0.}

and S1**Z2 <- {(N,N)}

Result: R - a complex number.

Range: R <- {(N,N)}

Error results:

Error Number	Arguments	Result
670216	S1 = +/-INDEF	(+IND, +IND)
670217	Z2 = +/-INDEF	(+IND, +IND)
670218	S1 = +/-INF	(+IND, +IND)
670219	Z2 = +/-INF	(+IND, +IND)
670220	S1 = 0.	
and	Re(Z2) < 0. or Im(Z2) =/ 0.	(+IND, +IND)
670221	S1**Z2 = +/-INF	(+IND, +IND)

2.8.80 ZTOD

Function: ZTOD

Description: Raise a complex base to a double precision power.

Entry points: call-by-reference MLP$RZTOD, ZTOD
call-by-value MLP$VZTOD

Arguments: Z1 - a complex number.
D2 - a double precision number.

Domain: Z1 <- {(N,N)}

and D2 <- {N}

and if Z1 = (0.,0.), D2 > 0.
and Z1**D2 <- {(N,N)}

Result: R - a complex number.

Range: R <- {(N,N)}

Error results:

Error Number	Arguments	Result
670222	Z1 = +/-INDEF	(+IND, +IND)
670223	D2 = +/-INDEF	(+IND, +IND)
670224	Z1 = +/-INF	(+IND, +IND)
670225	D2 = +/-INF	(+IND, +IND)
670226	Z1 = 0. and D2 < 0.	(+IND, +IND)
670227	Z1**D2 = +/-INF	(+IND, +IND)

2.8.81 ZTOI

Function: ZTOI

Description: Raise a complex base to an integer power.

Entry points: call-by-reference MLP$RZTOI, ZTOI
call-by-value MLP$VZTOI

Arguments: Z1 - a complex number.
I2 - an integer.

Domain: Z1 <- {(N,N)}

and I2 <- {all integers}

and Z1**I2 <- {(N,N)}

and if Z1 = (0.,0.), I2 > 0

Result: R - a complex number.

Range: R <- {(N,N)}

Error results:

Error Number	Arguments	Result
670228	Z1 = +/-INDEF	(+IND, +IND)
670229	Z1 = +/-INF	(+IND, +IND)
670230	Z1**I2 = +/-INF	(+IND, +IND)
670231	Z1 = 0. and I2 < 0	(+IND, +IND)

2.8.82 ZTOX

Function: ZTOX

Description: Raise a complex base to a single precision power.

Entry points: call-by-reference MLP$RZTOX, ZTOX
call-by-value MLP$VZTOX

Arguments: Z1 - a complex number.
S2 - a single precision number.

Domain: Z1 <- {(N,N)}
+
and S2 <- {N}
+
and if Z1 = (0.,0.), S2 > 0
and Z1**S2 <- {(N,N)}
+

Result: R - a complex number.

Range: R <- {(N,N)}
+

Error results:

+

Error Number	Arguments	Result
670232	Z1 = +/-INDEF	(+IND, +IND)
670233	S2 = +/-INDEF	(+IND, +IND)
670234	Z1 = +/-INF	(+IND, +IND)
670235	S2 = +/-INF	(+IND, +IND)
670236	Z1 = 0. and S2 < 0.	(+IND, +IND)
670237	Z1**S2 = +/-INF	(+IND, +IND)

2.8.83 ZTOZ

Function: ZTOZ

Description: Raise a complex base to a complex power.

Entry points:
call-by-reference MLP$RZTOZ, ZTOZ
call-by-value MLP$VZTOZ

Arguments:
Z1 - a complex number.
Z2 - a complex number.

Domain: Z1 <- {(N,N)}

and Z2 <- {(N,N)}

and if Z1 = (0.,0.), Z2 <- {(n1,n2) : n1 > 0., n2 = 0.}

and Z1**Z2 <- {(N,N)}

Result: R - a complex number.

Range: R <- {(N,N)}

Error results:

Error Number	Arguments	Result
670238	Z1 = +/-INDEF	(+IND, +IND)
670239	Z2 = +/-INDEF	(+IND, +IND)
670240	Z1 = +/-INF	(+IND, +IND)
670241	Z2 = +/-INF	(+IND, +IND)
670242	Z1 = 0	
and	Re(Z2) < 0. or Im(Z2) =/ 0.	(+IND, +IND)
670243	Z1**Z2 = +/-INF	(+IND, +IND)

3.0 NUMERIC CONVERSION AND ASSEMBLY LANGUAGE SUPPORT ROUTINES

3.1 INTRODUCTION

The CMML includes, in addition to the mathematical functions already described, a number of numeric conversion routines and assembly language support routines which will be referred to jointly as the CMML Common Support Routines. These routines are provided for all products (compiler or runtime systems) to perform numeric input and output conversion and other services and to allow code sharing. This will also ensure that the same numeric representation matches the same internal bit value by all processors. For performance purposes, the support routines are written in C180 assembly language.

The numeric conversion routines provide for the conversion between ASCII character strings and internal numeric representations. The assembly language support routines (formerly described in DCS document S3410) give the user access to some C180 hardware BDP and real arithmetic operations not readily available through CYBIL. The CMML support also provides some special conversion routines and capabilities specifically requested by the FMU project and other development organizations, because the improved performance of writing them directly in the C180 assembly language justified the abandonment of CYBIL for these procedures.

3.2 DOCUMENTATION CONVENTIONS

The naming convention for types, values, declarations, and procedures conform to the SIS naming conventions with the first two characters being 'ML' to indicate a Math Library (CMML) name. The third character indicates the type of name and the fourth character is a '$'.

The general linkage interface, error handling, and parameter type specifications for the common support routines are discussed in the following sections. The types and values used in the CMML support routines are presented as CYBIL declarations. Each support routine and its associated parameter list are described in CYBIL format in the specifications section by its XREF procedure declaration common deck.

3.3 LINKAGE INTERFACE

The linkage interface for the CMML support routines is defined in CYBIL terms and conforms to the CYBER 180 System Interface Standard (SIS) for inter-language procedure calls. The calling sequences are described in the routine specifications.

3.4 ERROR HANDLING

The CMML support routines are assembly language procedures designed so that no trap conditions are generated. There are no error numbers or messages associated with these routines. A status parameter whose MLT$ERROR value is returned to the caller indicates the quality of the result returned.

3.5 CONVERSION AND ALSS ROUTINE SPECIFICATIONS

This section contains procedure declarations with parameter list specifications and functional descriptions for the conversion and ALSS (Common Support) routines. Special CMML types, constants and values used in the descriptions are defined in Appendix A.

The meaning and usage of each parameter are usually obvious from its name and the context of the particular routine procedure. The most commonly used parameter names have the following meanings:

- Source — Pointer to the input source data to be processed.

- Source_length — Length of the source input (Units vary according to the routine).

- Target — Usually specifies the desired destination of the result. Sometimes it specifies an additional source parameter.

- Target_length — If this is a VAR parameter, the actual length of the result is returned in this parameter. Otherwise, on input, it specifies the desired length of the result.

- Status — An MLT$ERROR value is returned to caller via this parameter to indicate the quality of the result by specifying error status or special condition that occurred.

3.5.1 MLP$BDP_CONVERSION

```
{ MLD$BDP - Declare mlp$bdp_conversion }


PROCEDURE [XREF] mlp$bdp_conversion (source: ^cell;
      source_length: mlt$bdp_length;
      source_type: mlt$bdp_type;
      target: ^cell;
      target_length: mlt$bdp_length;
      target_type: mlt$bdp_type;
  VAR status: mlt$error);

{ FUNCTION: Provide access to the numeric move (MOVN) C180 hardware
{instruction.

{ STATUS MLE$INVALID_BDP_DATA is returned whenever the source or
{target type is mlc$alphanumeric, whenever invalid BDP data
{is contained in the source, or whenever a source or target
{length is inappropriate for its type.
{ STATUS MLE$LOSS_OF_SIGNIFICANCE is returned when the target field
{is not large enough to contain the converted source. The target
{will contain the rightmost significant digits of the converted
{source.
```

3.5.2 MLP$BDP_TO_BITS AND MLP$BITS_TO_BDP

```
{ MLD$BIT - Declare mlp$bdp_to_bits }
{      and - Declare mlp$bits_to_bdp }


PROCEDURE [XREF] mlp$bdp_to_bits (source: ^cell;
      source_length: mlt$bdp_length;
      source_type: mlt$bdp_type;
      target: ^cell;
      target_length: mlt$string_length;
      target_bit_offset: 0 .. 7;
  VAR negative: boolean;
  VAR status: mlt$error);

PROCEDURE [XREF] mlp$bits_to_bdp (source: ^cell;
      source_length: mlt$string_length;
      source_bit_offset: 0 .. 7;                                  :
      source_type: mlt$integer_type;                              :
      target: ^cell;
      target_length: mlt$bdp_length;
      target_type: mlt$bdp_type;
  VAR status: mlt$error);

{ FUNCTION: Convert a BDP number into an unaligned bit string (and
{vice versa). Written at the request of the FMU project.
{
{ In both procedures, the length of the bit string is in bits, not
{in bytes. The converted source is always placed right-justified   :
{in the target field with zero fill to the left unless the source  :
{in mlp$bits_to_bdp is signed and negative. All BDP types          :
{except alphanumeric are allowed.
{
{ NEGATIVE return a value of true whenever the source is negative.
{
{ STATUS MLE$BAD_PARAMETERS is returned whenever READ parameters are
{out of range.
```

```
{ STATUS MLE$LOSS_OF_SIGNIFICANCE is returned whenever the target is
{too small to contain the converted source. Truncation of the
{left-most digits occurs to force fit the result.
{ STATUS MLE$INVALID_BDP_DATA is returned whenever a source bdp
{number contains invalid characters.
```


3.5.3 MLP$COMPARE_BDP

```
{ MLD$CMN - Declare mlp$compare_bdp }


PROCEDURE [XREF] mlp$compare_bdp (source: ^cell;
      source_length: mlt$bdp_length;
      source_type: mlt$bdp_type;
      target: ^cell;
      target_length: mlt$bdp_length;
      target_type: mlt$bdp_type;
  VAR result: mlt$compare;
  VAR status: mlt$error);

{ FUNCTION: Provide access to the decimal compare (CMPN) C180
{hardware instruction. The user is referred to the MIGDS
{for information regarding the BDP types that are acceptable
{to this instruction.
{
{ STATUS MLE$INVALID_BDP_DATA is returned whenever BDP type or
{length is illegal for this hardware instruction.
```

3.5.4 MLP$COMPARE_BYTES

```
{ MLD$COM - Declare mlp$compare_bytes }


PROCEDURE [XREF] mlp$compare_bytes (source: ^cell;
      source_length: mlt$string_length;
      target: ^cell;
      target_length: mlt$string_length;
  VAR result: mlt$compare;
  VAR number_equal_bytes: mlt$string_length;
  VAR status: mlt$error);

{ FUNCTION: Provide access to the compare bytes (CMPB) C180
{instruction without limiting the user to byte lengths less
{than or equal to 256.
{
{ STATUS MLE$NO_ERROR will be returned.
```

3.5.5 MLP$COMPARE_COLLATED

```
{ MLD$CCI - Declare mlp$compare_collated }


PROCEDURE [XREF] mlp$compare_collated (source: ^cell;
      source_length: mlt$string_length;
      target: ^cell;
      target_length: mlt$string_length;
      collate_table: ^cell;
  VAR result: mlt$compare;
  VAR number_equivalent_bytes: mlt$string_length;
  VAR status: mlt$error);

{ FUNCTION: Provide access to the compare collated (CMPC) C180
{hardware instruction without restricting the user to byte
{lengths less than or equal to 256.
{
{ STATUS MLE$NO_ERROR is returned.
```

3.5.6 MLP$COMPARE_FLOATING

```
{ MLD$CF - Declare mlp$compare_floating }


PROCEDURE [XREF] mlp$compare_floating (source: ^cell;
      source_length: mlt$floating_length;
      target: ^cell;
      target_length: mlt$floating_length;
  VAR result: mlt$compare;
  VAR status: mlt$error);

{ FUNCTION: Compare the values of two floating point numbers.
{
{ STATUS MLE$INDEFINITE is returned whenever the source or target is
{indefinite or whenever both source and target are infinite with the
{same sign. The result is then MLC$UNORDERED.
```

3.5.7 MLP$COMPUTE_FLOATING_NUMBER

```
{ MLD$CFN - Declare mlp$compute_floating_number }


PROCEDURE [XREF] mlp$compute_floating_number (source:
  mlt$floating_input;
      scale_factor: integer;
      target: ^cell;
      target_length: mlt$floating_length;
  VAR status: mlt$error);

{ FUNCTION: Generate an internal (binary) floating point number
{given as input a scale factor (power of ten) and the TARGET
{parameter result of MLP$INPUT_FLOATING_MANTISSA (as SOURCE).
{
{ STATUS MLE$OVERFLOW is returned whenever the floating point number
{"generated" is out of range (that is - infinite or indefinite).
{ The value returned will be either +INF or +IND, depending on the
{nature of the overflow.
```

3.5.8 MLP$CONVERT_FLOAT_TO_INTEGER

```
{ MLD$CFI - Declare mlp$convert_float_to_integer }


PROCEDURE [XREF] mlp$convert_float_to_integer (source: ^cell;
      source_length: mlt$floating_length;
      target: ^cell;
      target_length: mlt$integer_length;
      target_type: mlt$integer_type;
  VAR status: mlt$error);

{ FUNCTION: Convert a floating point number into an integer.
{
{ STATUS MLE$LOSS_OF_SIGNIFICANCE is returned whenever the floating
{point number cannot be represented as an integer of the specified
{length. The integer value returned will contain the rightmost
{significant bits of the correct result. For infinite or indefinite
{floating point numbers, the integer value returned is 0.
```

3.5.9 MLP$CONVERT_INTEGER_TO_FLOAT

```
{ MLD$CIF - Declare mlp$convert_integer_to_float }

PROCEDURE [XREF] mlp$convert_integer_to_float (source: ^cell;
      source_length: mlt$integer_length;
      source_type: mlt$integer_type;
      target: ^cell;
      target_length: mlt$floating_length;
  VAR status: mlt$error);

{ FUNCTION: Convert an integer into a floating point number.
{
{ STATUS MLE$NO_ERROR is returned.
```

3.5.10 MLP$INPUT_BASE_NUMBER

```
{ MLD$IBN - Declare mlp$input_base_number }


PROCEDURE [XREF] mlp$input_base_number (source: ^cell;
      source_length: mlt$string_length;
      target: ^cell;
      target_length: mlt$string_length;
      base: mlt$non_decimal_base;
      inbedded_blanks: mlt$handle_blanks;
      justification: mlt$justify;
  VAR actual_source_length: mlt$string_length;
  VAR status: mlt$error);

{ FUNCTION: Convert an ASCII representation of a non-decimal base
{number into an internal binary representation. Leading ASCII
{blanks are ignored; leading ASCII zeroes will be converted as part
{of the number. The ASCII number is considered to be unsigned.
{
{ The TARGET_LENGTH is in bytes.
{
{ The ACTUAL_SOURCE_LENGTH returned is the number of source
{characters processed, including leading blanks and blanks that were
{ignored or treated as zeros. Illegal characters and blanks treated
{as illegal (MLC$STOP_ON_BLANKS) are not included in the actual
{length.
{
{ STATUS MLE$BAD_PARAMETERS is returned whenever READ parameters are
{out of range.
{ STATUS MLE$LOSS_OF_SIGNIFICANCE occurs when the target field is
{too small to contain the converted source. The rightmost
{significant bits are truncated in the target field.
{ STATUS MLE$INVALID_BDP_DATA is returned when an illegal "digit" is
{present in the source field. A terminating blank or comma is NOT
{considered illegal. The input field to that point will be
{converted.
```

3.5.11 MLP$INPUT_FLOATING_MANTISSA

```
{ MLD$IFM - Declare mlp$input_floating_mantissa }


PROCEDURE [XREF] mlp$input_floating_mantissa (source: ^cell;
      source_length: mlt$string_length;
      imbedded_blanks: mlt$handle_blanks;
  VAR target: mlt$floating_input;
  VAR decimal_point_found: boolean;
  VAR actual_source_length: mlt$string_length;
  VAR status: mlt$error);

{ FUNCTION: Convert an ASCII representation of a floating point
{mantissa into an internal representation for later conversion to
{internal floating point after establishing the value of the
{exponent field. Leading blanks and zeroes are ignored.
{
{ STATUS MLE$BAD_PARAMETERS is returned whenever READ parameters are
{out of range.
{ STATUS MLE$INVALID_BDP_DATA is returned whenever an illegal
{character is detected in the source. This situation includes
{possible exponent field characters "E" and "D", completely blank
{fields, and source fields containing only a sign character. In the
{latter two cases, the field is considered to be identically zero. A
{terminating blank or comma is NOT considered illegal.
```

3.5.12 MLP$INPUT_FLOATING_NUMBER

```
{ MLD$IFN - Declare mlp$input_floating_number }


PROCEDURE [XREF] mlp$input_floating_number (source: ^cell;
      source_length: mlt$string_length;
      target: ^cell;
      target_length: mlt$floating_length;
      handle_blanks: mlt$handle_blanks;
  VAR actual_source_length: mlt$string_length;
  VAR status: mlt$error);

{ FUNCTION: Convert an ASCII representation of a floating point
{number (with an optional exponent field) into the internal
{(binary) floating point representation.
{
{ RESTRICTIONS: The exponent field must begin with "E", "D", "e",
{or "d". Arithmetic overflow during exponent computation is ignored.
{
{ The only valid values for the HANDLE_BLANKS parameter are
{MLC$IGNORE_BLANKS and MLC$STOP_ON_BLANK.
{
{ STATUS MLE$INVALID_BDP_DATA is returned whenever an illegal
{character is detected in the source field. A terminating blank or
{comma is NOT considered illegal.
{ STATUS MLE$OVERFLOW will be returned whenever the floating point
{number is infinite or indefinite AND status is otherwise no error.
{ STATUS MLE$NO_DIGITS is returned if no digits were found in the
{source.
```

3.5.13 MLP$INPUT_INTEGER

```
{ MLD$II - Declare mlp$input_integer }

PROCEDURE [XREF] mlp$input_integer (source: ^cell;
      source_length: mlt$string_length;
      target: ^cell;
      target_length: mlt$integer_length;
      target_type: mlt$integer_type;
      imbedded_blanks: mlt$handle_blanks;
  VAR actual_source_length: mlt$string_length;
  VAR status: mlt$error);

{ FUNCTION: Convert an ASCII representation of an integer into the
{internal (binary) representation.
{
{ STATUS MLE$NO_DIGITS is returned whenever the source string
{contains no digits (ASCII characters in the set '0'...'9').
{ STATUS MLE$INVALID_BDP_DATA is returned whenever an illegal
{character is detected in the source field. A blank does NOT cause
{this error status. STATUS MLE$LOSS_OF_SIGNIFICANCE is returned
{whenever the internal integer field is too small to contain the
{converted ASCII source. The rightmost significant bits are
{retained.
```

3.5.14 MLP$INPUT_UNPACKED_DECIMAL

```
{ MLD$IUD - Declare mlp$input_unpacked_decimal }

PROCEDURE [XREF] mlp$input_unpacked_decimal (source: ^cell;
      source_length: mlt$string_length;
      target: ^cell;
      target_length: mlt$bdp_length;
  VAR actual_source_length: mlt$string_length;
  VAR status: mlt$error);

{ FUNCTION: Convert an ASCII representation of an unpacked decimal
{number (with possibly leading blanks and/or a leading sign) into
{the internal BDP format of UNPACKED DECIMAL TRAILING SIGN
{COMBINED HOLLERITH. The result will be right justified in the
{target field. If the result is shorter than the target field, the
{target field will be zero filled to the left. The final digit will
{be changed to conform to the preferred combined sign format.
{Written at the request of the COBOL and FMU projects.
{
{ If a decimal point is encountered before the source field is
{exhausted, it terminates the source input and only the digits
{preceding the decimal point are converted. The decimal point is
{counted in the actual_source_length returned and is not considered
{an illegal character.
{
{ STATUS MLE$INVALID_BDP_DATA is returned whenever an illegal
{character is detected in the source. The source is converted up to
{the illegal character. The illegal character is not counted in the
{actual_source_length returned.
{ STATUS MLE$LOSS_OF_SIGNIFICANCE is returned whenever the target
{field is too small to contain the source number. The rightmost
{significant digits are retained. Also, if the length of the
{significant digits of the source, including the optional sign,
{exceeds 38 bytes, STATUS MLE$LOSS_OF_SIGNIFICANCE is returned. Only
{the first 38 bytes from the left will be converted. The
{actual_source_length returned will include a count of all
{significant digits encountered in the source even though not all
{will be converted.
```

3.5.15 MLP$MOVE_BYTES

```
{ MLD$MOV - Declare mlp$move_bytes }


PROCEDURE [XREF] mlp$move_bytes (source: ^cell;
      source_length: mlt$string_length;
      target: ^cell;
      target_length: mlt$string_length;
  VAR status: mlt$error);

{ FUNCTION: Provide access to move bytes (MOVB) C180 hardware
{instruction without restricting the caller to fields less than or
{equal to 256 bytes. Furthermore, allow overlapping source and
{target fields.
{
{ STATUS will be MLE$NO_ERROR
```

3.5.16 MLP$OUTPUT_BASE_NUMBER

```
{ MLD$OBN - Declaration of mlp$output_base_number }


PROCEDURE [XREF] mlp$output_base_number (source: ^cell;
      source_length: mlt$string_length;
      target: ^cell;
      target_length: mlt$string_length;
      base: mlt$non_decimal_base;
      justification: mlt$justify;
      suppress_leading_zeros: boolean;
  VAR actual_target_length: mlt$string_length;
  VAR status: mlt$error);

{ FUNCTION: Convert a binary integer into an (non-decimal) ASCII
{representation, or simply do a memory dump.
{
{ SOURCE_LENGTH is in bytes.
{
{ All bytes of the source number are converted and may yield
{leading zeros which are part of the converted number. These
{zeros may be suppressed in the target by setting parameter
{SUPPRESS_LEADING_ZEROS to the value TRUE.
{
{ When the target_length (including leading zeros, if any) is
{less than the size of the target area, blanks may be used to
{fill in the rest of the area.
{
{ When JUSTIFICATION is MLC$RIGHT_JUSTIFY, blank fill is used. For
{MLC$LEFT_JUSTIFY, no fill is done.
{
{ ACTUAL_TARGET_LENGTH is the number of non-blank ASCII characters
{written to the target.
{
{ STATUS MLE$LOSS_OF_SIGNIFICANCE is returned whenever the target
{field is too small to contain the converted source. Truncation of
{digits at the left occurs for right justification. Truncation at
{the right occurs for left justification.
```

3.5.17 MLP$OUTPUT_FLOATING_DIGITS

```
{ MLD$OFD - Declare mlp$output_floating_digits }


PROCEDURE [XREF] mlp$output_floating_digits (source: ^cell;
      source_length: mlt$string_length;
      target: ^cell;
      target_length: mlt$string_length;
      leading_blanks: mlt$string_length;
      leading_zeroes: mlt$string_length;
      decimal_point: mlt$string_length;
      sign_character: char;
  VAR status: mlt$error);

{ FUNCTION: Generate an ASCII floating point mantissa given an ASCII
{or unpacked decimal trailing sign combined hollerith string of
{digits and formatting information.
{
{ The value of DECIMAL_POINT is the location in the target "string"
{of the decimal point character. Note that the first position in the
{string has an index of 0.
{
{ TARGET_LENGTH must be greater than SOURCE_LENGTH + LEADING_BLANKS
{+ ord( SIGN_CHARACTER <> chr( 0 ) ).
{
{ The target area will be right-filled with zeroes if necessary to
{entirely fill the field.
{
{ STATUS will contain MLE$NO_ERRORR.
```

3.5.18 MLP$OUTPUT_FLOATING_NUMBER

```
{ MLD$OFN - Declare mlp$output_floating_number }


PROCEDURE [XREF] mlp$output_floating_number (source: ^cell;
      source_length: mlt$floating_length;
      target: ^cell;
      format: mlt$output_format;
  VAR actual_target_length: mlt$string_length;
  VAR status: mlt$error);

{ FUNCTION: Convert a floating point number into an ASCII
{representation.
{
{ FORMAT describes the format of the result string. The names of the
{ordinals for the FORMAT field (of the same-named parameter) are
{derived from FORTRAN-style format descriptors.
{ When the FORMAT field contains MLC$LIST_DIRECTED, the number is
{output in either a modified E or modified F format. If the absolute
{value of the number is greater than or equal to 10**-6 and less
{than 10**9, the modified F format is used; otherwise the modified E
{format is used. The DIGITS field gives the number of digits to
{which the number is rounded. Trailing zeroes after the decimal
{point are always removed. The SCALE_FACTOR field is ignored;
{rather, a scale_factor of 0 is used for the modified F style, and 1
{is used for the modified E format. The EXPONENT_STYLE field is also
{ignored. No exponent occurs for F style, and, for F style, the
{width of the field will be the minimum needed. If the WIDTH field
{is insufficient to hold the representation with all DIGITS
{significant digits, then digits will be truncated from the right of
{the mantissa in order to fit the representation into WIDTH
{characters.
{ When the FORMAT field does not contain MLC$LIST_DIRECTED, the
{EXPONENT_STYLE field contains either 0 or the number of digits in
{the exponent. When 0 is provided, the normal FORTRAN style of four
{characters for the exponent is used. When the JUSTIFICATION field
{indicates right justification, blank fill will occur on the left.
{Otherwise there is no fill.
{
{ ACTUAL_TARGET_LENGTH will contain the number of characters written
{to the target area, excluding any padding.
{
{ STATUS MLE$BAD_PARAMETERS is returned when FORMAT.WIDTH is
{inconsistent with the other fields of FORMAT, independent of the
{value of the floating point number.
{ STATUS MLE$INFINITE is returned whenever the source floating point
```

```
{number is infinite.
{ STATUS MLE$INDEFINITE is returned whenever the source floating
{point number is indefinite.
{ STATUS MLE$LOSS_OF_SIGNIFICANCE is returned whenever the
{particular value of the floating point number is not representable
{in the format specified.
```

3.5.19 MLP$OUTPUT_INTEGER

```
{ MLD$OI - Declare mlp$output_integer }

PROCEDURE [XREF] mlp$output_integer (source: ^cell;
      source_length: mlt$integer_length;
      source_type: mlt$integer_type;
      target: ^cell;
      target_length: mlt$string_length;
      justification: mlt$justify;
      sign: mlt$sign_treatment;
  VAR actual_target_length: mlt$string_length;
  VAR status: mlt$error);

{ FUNCTION: Convert an integer into an ASCII representation.
{
{ When JUSTIFICATION is MLC$RIGHT_JUSTIFY, the target area is
{blank-filled to the left. Otherwise no fill is done.
{
{ ACTUAL_TARGET_LENGTH will contain the number of digits written to
{the target area plus 1, if there is a sign.
{
{ STATUS MLE$LOSS_OF_SIGNIFICANCE is returned whenever the target
{field is too small to contain the converted source. Truncation of
{the leftmost digits occurs.
```

3.5.20 MLP$ROUND_FLOATING_NUMBER

```
{ MLD$RFN - Declare mlp$round_floating_number }


PROCEDURE [XREF] mlp$round_floating_number (source: ^cell;
      source_length: mlt$floating_length;
      target: ^cell;
      number_of_digits: mlt$digit_string_length;
      power_of_ten: integer;
  VAR status: mlt$error);

{ FUNCTION: Convert a floating point number into an ASCII string
{containing the first NUMBER_OF_DIGITS significant digits (rounded).
{MLP$SCALE_FLOATING_NUMBER must be called before
{MLP$ROUND_FLOATING_NUMBER, and the POWER_OF_TEN result of
{MLP$SCALE_FLOATING_NUMBER must be passed to
{MLP$ROUND_FLOATING_NUMBER.
{
{ MLP$ROUND_FLOATING_OUTPUT and MLP$SCALE_FLOATING_OUTPUT must be
{used by all C180 products for the output of floating point
{numbers to ensure uniform representation throughout the C180
{product set. MLP$OUTPUT_FLOATING_NUMBER will do this for the user,
{provided that the available floating point formats of the latter
{procedure are adequate for the user's purpose.
{
{ STATUS MLE$BAD_PARAMETERS is returned whenever the floating point
{number is infinite or indefinite. (This should have been caught
{by the call to MLP$SCALE_FLOATING_NUMBER.)
{ STATUS MLE$OVERFLOW is returned whenever the rounded source
{number's POWER_OF_TEN differs from the actual power as passed by
{the caller. The digit string returned is then "10...0".
```

3.5.21 MLP$SCALE_FLOATING_NUMBER

```
{ MLD$SFN - Declare mlp$scale_floating_number }

PROCEDURE [XREF] mlp$scale_floating_number (source: ^cell;
      source_length: mlt$floating_length;
  VAR power_of_ten: integer;
  VAR status: mlt$error);

{ FUNCTION : Determine the value of the (decimal) exponent of a
{floating point number in the form d.dd ... E ..
{
{ POWER_OF_TEN will contain 0 if the floating point number is zero.
{Otherwise, if x is the absolute value of the floating point number
{and 1.0 <= x * 10**e < 10.0, then POWER_OF_TEN will contain e.
{
{ STATUS MLE$INDEFINITE is returned whenver the source is
{indefinite. STATUS MLE$INFINITE is returned whenever the source is
{infinite.
```

3.5.22 MLP$SCAN_BYTES

```
{ MLD$SCA - Declare mlp$scan_bytes }


PROCEDURE [XREF] mlp$scan_bytes (source: ^cell;
      source_length: mlt$string_length;
      scan_table: ^cell;
  VAR number_not_matching: mlt$string_length;
  VAR status: mlt$error);

{ FUNCTION: Provide access to the scan bytes while non-member (SCNB)
{C180 hardware instruction, without restricting the caller to
{lengths less than or equal to 256 bytes.
{
{ STATUS will contain MLE$NO_ERROR.
```


3.5.23 MLP$TEST_FOR_EXCEPTION

```
{ MLD$TEX - Declare mlp$test_for_exception }


PROCEDURE [XREF] mlp$test_for_exception (source: ^cell;
  VAR status: mlt$error);

{ FUNCTION: Test a floating point number for infinite or indefinite.
{
{ If the number is indefinite, return MLE$INDEFINITE in STATUS.
{ If the number is infinite, return MLE$INFINITE. Otherwise
{return MLE$NO_ERROR.
```


3.5.24 MLP$TRANSLATE_BYTES

```
{ MLD$TRA -- Declare mlp$translate_bytes }


PROCEDURE [XREF] mlp$translate_bytes (source: ^cell;
      source_length: mlt$string_length;
      target: ^cell;
      target_length: mlt$string_length;
      translation_table: ^cell;
  VAR status: mlt$error);

{ FUNCTION: Provide access to the translate bytes (TRANB) C180
{hardware instruction without restricting the source or target to
{a maximum of 256 bytes.
{
{ STATUS will always be MLE$NO_ERROR.
```


3.5.25 MLP$VAX_TO_180_FLOATING

```
{ MLDVAXF -- Declare mlp$VAX_to_180_floating }


PROCEDURE [XREF] mlp$vax_to_180_floating (source: ^cell;
      source_type: mlt$vax_floating_type;
      target: ^cell;
      target_length: mlt$floating_length;
  VAR status: mlt$error);

{ FUNCTION: Convert a VAX floating point number of the specified
{source_type to a C180 floating point number of the specified
{target length.
{
{
{ LENGTH AND SIZE INFORMATION FOR FLOATING TYPES:
{
{      TYPE                        LENGTH     EXPONENT     TRUE FRACTION
{                                  (BYTES)    SIZE (BITS)   SIZE (BITS)
{ -----------------                -------    -----------  -------------
{
{
{ mlc$vax_4_F_float                  4            8             24
{ mlc$vax_8_D_float                  8            8             56
{ mlc$vax_8_G_float                  8           11             53
{ mlc$vax_16_H_float                16           15            113
{ mlc$single_precision               8           15             48
{ mlc$double_precision              16           15             96
{
{
{ ERROR STATUS:
{MLE$BAD_PARAMETERS is returned whenever source_type or target_
{length is out-of-range.
{
{All VAX Reserved Operand values are converted to C180 +INFINITE
{and status MLE$INFINITE is returned.
{
{No other errors can occur for mlc$vax_4_f_float type conversion.
{Such values can always be converted exactly to C180 floating
{point formats.
{
{Mlc$vax_8_d_float and mlc$vax_8_g_float VAX values can always be
{represented within range in C180 format regardless of target
{length. However, significance can be lost as a result of the
{fewer number of fraction bits available for C180 single_precision
{floating point format. The result is rounded to 48 bits of
```



```
{significance and MLE$LOSS_OF_SIGNIFICANCE is returned. Signif-
{icance can be preserved for these VAX 8-byte types by specifying
{C180 mlc$double_precision for the target_length.
{
```

```
{VAX mlc$vax_16_H_float values can exceed C180 double precision
{values in both range and precision. Since there is such a large
{difference in the number of fraction bits between the VAX and
{C 180 16-byte floating point formats, the result is rounded to
{96 bits of precision, but no loss_of_significance error will be
{signaled for these conversions unless the target length was
{specified as mlc$single_precision.
{
{The table below shows the result and error status for VAX values
{that are out-of-range for C180 single and double precision
{floating point numbers. VAX values that convert to C180 values
{with the following C180 biased exponents will produce the
{indicated results. The exponents include the sign bit:
{
{
{  C180 BIASED EXPONENT          RESULT          ERROR STATUS
{ ----------------------       ----------       ------------
{
{
{ 0XXX or 8XXX                     0            MLE$NO_ERROR
{
{ 1000-2FFF or 9000-AFFF           0            MLE$UNDERFLOW
{
{ 5000-6FFF                    +INFINITE        MLE$OVERFLOW
{
{ D000-EFFF                    -INFINITE        MLE$OVERFLOW
{
{ 7XXX                         +INDEFINITE      MLE$INDEFINITE
{
{ FXXX                         -INDEFINITE      MLE$INDEFINITE
{
{ *VAX Reserved Operand*       +INFINITE        MLE$INFINITE
{
{
```


3.5.26 MLP$VAX_TO_180_FORTRAN_LOGICAL

```
{ MLDVAXL -- Declare mlp$VAX_to_180_fortran_logical }
```

```
PROCEDURE [XREF] mlp$vax_to_180_fortran_logical (source: ^cell;
      source_length: mlt$vax_logical_length;
      target: ^cell;
      target_length: mlt$FORTRAN_logical_length;
  VAR status: mlt$error);

{ FUNCTION: Convert a VAX logical value to a C180 FORTRAN
{logical value of the specified length. The right most bit in
{the first byte of the VAX value is used to determine the
{logical value. A one bit means TRUE and a zero in this bit
{means FALSE. The C180 FORTRAN logical result uses the sign
{bit (bit 0) of the result to indicate its logical value.
{The sign bit of the target will be set to a one for TRUE
{and to a zero for FALSE. The remaining bits in the result
{will be all zeros.
{
{ ERROR STATUS:
{MLE$BAD_PARAMETERS is returned whenever source_length or
{target_length is out-of-range; otherwise, STATUS will always
{be MLE$NO_ERROR.
{
```


3.5.27 MLP$VAX_TO_180_INTEGER

```
{ MLDVAXI -- Declare mlp$VAX_to_180_integer }
```

```
PROCEDURE [XREF] mlp$vax_to_180_integer (source: ^cell;
      source_length: mlt$vax_integer_length;
      target: ^cell;
      target_length: mlt$integer_length;
  VAR status: mlt$error);

{ FUNCTION: Convert a two's complement signed integer value in
{VAX format to a signed integer in C180 format. The target result
{is always right-justified with sign extension to the left.
{
{ ERROR STATUS:
{MLE$BAD_PARAMETERS is returned whenever the source_length or the
{target_length is out-of-range.
{
{MLE$LOSS_OF_SIGNIFICANCE is returned when the VAX number is not
{representable as a C180 number of the specified length. The C180
{result is truncated at the left to fit the target field.
{
```


3.5.28 MLP$VAX_TO_180_PACKED_DECIMAL

```
{ MLDVAXPD -- Declare mlp$VAX_to_180_packed_decimal }
```

```
PROCEDURE [XREF] mlp$vax_to_180_packed_decimal (source: ^cell;
      source_length: mlt$vax_packed_decimal_length;
      target: ^cell;
      target_length: mlt$bdp_length;
  VAR status: mlt$error);

{ FUNCTION: Convert a VAX packed decimal value of the specified
{length to a C180 packed decimal value of the desired target_length.
{
{ ERROR STATUS:
{STATUS MLE$BAD_PARAMETERS is returned whenever the source_length
{or target_length is out-of-range.
{
{STATUS MLE$LOSS_OF_SIGNIFICANCE is returned whenever the target
{field is too small to contain the converted source. The target
{will contain the rightmost significant digits of the converted
{source.
{
```


3.5.29 MLP$170_TO_180_BINARY

```
{ MLD$788 - Declare mlp$170_to_180_binary }


PROCEDURE [XREF] mlp$170_to_180_binary (source: ^cell;
```

```
      source_length: mlt$string_length;
      source_bit_offset: 2 .. 7;
      target: ^cell;
      target_length: mlt$string_length;
      target_bit_offset: 0 .. 7;
  VAR status: mlt$error);

{ FUNCTION : convert a C170 bit string (in 6 of 8 format) into a
{C180 bit string. Written at the request of the FMU project.
{
{ Note that both source and target length are given in bits.
{
{ When the source_length is greater than the target_length, the
{target field is filled with the leftmost bits of the source with
{no error status returned.
{
{ When target_length is greater than source_length the target is
{right filled with zeroes.
{
{ STATUS MLE$BAD_PARAMETERS is returned when read-only parameters
{are out of range.
```


3.5.30 MLP$170_TO_180_FLOATING

```
{ MLD$78F - Declare mlp$170_to_180_floating }

PROCEDURE [XREF] mlp$170_to_180_floating (source: ^cell;
      target: ^cell;
```

```
      size: mlt$floating_length;
  VAR status: mlt$error);

{ FUNCTION: Convert a floating point number in C170 notation (6 of 8
{format) to a C180 floating point number. Written at the request of
{the FMU project.
{
{ STATUS MLE$BAD_PARAMETERS is returned whenever size is out of
{range.
{ STATUS MLE$INFINITE is returned when the C170 number has the
{exponent 3777(8) or 4000(8); the C180 value returned is +/- INF.
{ STATUS MLE$INDEFINITE is returned when the C170 number has the
{exponent 1777(8) or 6000(8); the C180 value returned is +/- INDEF.
```


3.5.31 MLP$170_TO_180_INTEGER

```
{ MLD$78I - Declare mlp$170_180_integer }

PROCEDURE [XREF] mlp$170_to_180_integer (source: ^cell;
      source_length: 1 .. 10;
      target: ^cell;
```

```
       target_length: mlt$integer_length;
       target_type: mlt$integer_type;
  VAR status: mlt$error);

{ FUNCTION: Convert an integer in C170 6 of 8 format to an integer
{in C180 format. The target is always right-justified with sign
{extension to the left.
{
{ C170 negative zero is represented as zero (0.0) on the C180.
{
{ STATUS MLE$BAD_PARAMETERS is returned whenever a read-only
{parameter is out-of-range.
{ STATUS MLE$LOSS_OF_SIGNIFICANCE is returned when the C170 number
{is not representable as a C180 number of the specified length
{and type. Truncation at the left occurs to force-fit the
{remainder.
```


3.5.32 MLP$180_TO_170_BINARY

```
{ MLD$87B - Declare mlp$180_to_170_binary }

PROCEDURE [XREF] mlp$180_to_170_binary (source: ^cell;
       source_length: mlt$string_length;
       source_bit_offset: 0 .. 7;
       target: ^cell;
```

```
      target_length: mlt$string_length;
      target_bit_offset: 2 .. 7;
  VAR status: mlt$error);

{ FUNCTION: Convert C180 bit strings (non-aligned) into C170 bit
{strings (also non-aligned) in 6 of 8 format. Written at the
{request of the FMU project.
{
{ Note that both SOURCE_LENGTH and TARGET_LENGTH are in bits.
{
{ When TARGET_LENGTH is greater than SOURCE_LENGTH, the target is
{right filled with zeroes.
{
{ When SOURCE_LENGTH is greater than TARGET_LENGTH, the target is
{filled with the leftmost bits of the source. No error status is
{recorded.
{
{ STATUS MLE$BAD_PARAMETERS is returned whenever a READ only
{parameter is out-of-range.
```


3.5.33 MLP$180_TO_170_FLOATING

```
{ MLD$87F - Declare mlp$180_to_170_floating }

PROCEDURE [XREF] mlp$180_to_170_floating (source: ^cell;
      target: ^cell;
      size: mlt$floating_length;
  VAR status: mlt$error);
```

```
{ FUNCTION: Convert a C180 floating point number into a C170
{floating point number (in 6 of 8 format). Written at the
{request of the FMU project.
{
{ STATUS MLE$BAD_PARAMETERS is returned if size is out of range.
{ STATUS MLE$UNDERFLOW is returned when the C180 exponent is too
{small to be represented in C170 format. Zero is returned as the
{value of the C170 number.
{ STATUS MLE$OVERFLOW is returned when the C180 exponent is too
{large to be represented in C170 format. The C170 value returned
{in the case is 37770000000000000000(8), or 40000000000000000000(8)
{if the C180 number is negative.
{ STATUS MLE$INFINITE is returned whenever the C180 number is +/-
{INF. The C170 number returned will be 37770..0(8) or 4000..0(8),
{respectively.
{ STATUS MLE$INDEFINITE is returned whenever the C180 number is +/-
{INDEF. The C170 number returned will be 17770..0(8) or 6000..0(8),
{respectively.
```


3.5.34 MLP$180_TO_170_INTEGER

```
{ MLD$87I - Declare mlp$180_to_170_integer }


PROCEDURE [XREF] mlp$180_to_170_integer (source: ^cell;
      source_length: mlt$integer_length;
      source_type: mlt$integer_type;
      target: ^cell;
      target_length: 1 .. 10;
  VAR status: mlt$error);
```

```
{ FUNCTION: Convert an integer in C180 format into an integer in
{C170 format (6 of 8). The target field is always right-justified
{with sign extension on the left. Written at the request of the FMU
{project.
{
{ STATUS MLE$BAD_PARAMETERS is returned whenever a read-only
{parameter is out-of-range.
{ STATUS MLE$LOSS_OF_SIGNIFICANCE is returned whenever the C180
{number is not representable in the specified C170 format.
{Truncation occurs at the left of the source to force fit the
{remainder.
```


A1.0 TYPES AND CONSTANTS FOR SUPPORT ROUTINES

The CMML-defined types and constants used in the Common Support routines and their specifications are described here as CYBIL declarations.

A1.1 MLT$BDP_LENGTH

```
{ MLTBDPL -- Declaration of mlt$bdp_length }


CONST
  mlc$min_bdp_length = 0,
  mlc$max_bdp_length = 38;

TYPE
  mlt$bdp_length = mlc$min_bdp_length .. mlc$max_bdp_length;
```

A1.2 MLT$BDP_TYPE

+ ------------

```
{ MLTBDP -- Declaration of mlt$bdp_type }


TYPE
  mlt$bdp_type = (mlc$packed_unsigned, mlc$packed_unsigned_slack,
    mlc$packed_decimal_signed, mlc$packed_decimal_signed_slack,
    mlc$unpacked_unsigned, mlc$unpacked_trailing_hollerith,
    mlc$unpacked_trailing_separate, mlc$unpacked_leading_hollerith,
    mlc$unpacked_leading_separate, mlc$alphanumeric,
    mlc$binary_unsigned, mlc$binary_signed,
    mlc$translated_packed_signed, mlc$translated_packed_slack,
    mlc$translated_binary_unsigned, mlc$translated_binary_signed);
```

A1.3 MLT$COMPARE

+ ----------

```
{ MLTCOMP -- Declaration of mlt$compare }


TYPE
  mlt$compare = (mlc$equal, mlc$source_is_greater, mlc$unordered,
    mlc$target_is_greater);
```


A1.4 MLT$DIGIT_STRING_LENGTH

+ ----------------------

```
{ MLTDSL -- Declaration of mlt$digit_string_length }


CONST
  mlc$min_digit_string_length = 0,
```

```
    mlc$max_digit_string_length = 35;

  TYPE
    mlt$digit_string_length = mlc$min_digit_string_length ..
      mlc$max_digit_string_length;
```

A1.5 MLT$ERROR

\+ ---------

```
  { MLTERR -- Declaration of mlt$error }


  TYPE
    mlt$error = (mle$no_error, mle$invalid_bdp_data,
      mle$loss_of_significance, mle$overflow, mle$underflow,
      mle$indefinite, mle$infinite, mle$bad_parameters,
      mle$no_digits);
```

A1.6 MLT$EXPONENT_STYLE

\+ -----------------

```
  { MLTES -- Declaration of mlt$exponent_style }


  CONST
    mlc$min_exponent_style = 0,
    mlc$max_exponent_style = 6;

  TYPE
    mlt$exponent_style = mlc$min_exponent_style ..
      mlc$max_exponent_style;
```

A1.7 MLT$FLOATING_INPUT

\+ -----------------

```
  { MLTFI -- Declaration of mlt$floating_input }
```



```
  TYPE                                                            :
    mlt$floating_input = array [1 .. 120] of cell;                :
```

A1.8 MLT$FLOATING_LENGTH

\+ ------------------

```
{ MLTFL -- Declaration of mlt$floating_length }


TYPE
  mlt$floating_length = (mlc$single_precision,
    mlc$double_precision);
```

A1.9 MLT$FORMAT

```
{ MLTFORM -- Declaration of mlt$format }


TYPE
  mlt$format = (mlc$f_style, mlc$e_style, mlc$g_style,
    mlc$list_directed, mlc$namelist);
```

A1.10 MLT$FORTRAN_LOGICAL_LENGTH

```
{ MLTFTLL -- Declaration of mlt$fortran_logical_length }


TYPE
  mlt$fortran_logical_length = 1 .. 8;
```

A1.11 MLT$HANDLE_BLANKS

```
{ MLTHB -- Declaration of mlt$handle_blanks }


TYPE
  mlt$handle_blanks = (mlc$ignore_blanks, mlc$stop_on_blank,
    mlc$blanks_equal_zero);
```


A1.12 MLT$INTEGER_LENGTH

```
{ MLTIL -- Declaration of mlt$integer_length }


CONST
  mlc$min_integer_length = 1,
  mlc$max_integer_length = 8;

TYPE
  mlt$integer_length = mlc$min_integer_length ..
    mlc$max_integer_length;
```

A1.13 MLT$INTEGER_TYPE

+ ----------------

```
{ MLTIT -- Declaration of mlt$integer_type }


TYPE
  mlt$integer_type = (mlc$signed_integer, mlc$unsigned_integer);
```

A1.14 MLT$JUSTIFY

+ -----------

```
{ MLTJUST -- Declaration of mlt$justify }


TYPE
  mlt$justify = (mlc$left_justify, mlc$right_justify);
```

A1.15 MLT$NON_DECIMAL_BASE

+ --------------------

```
{ MLTNDB -- Type declarations for numeric conversion routines }


TYPE
  mlt$non_decimal_base = (mlc$binary, mlc$octal, mlc$hexadecimal);
```


A1.16 MLT$OUTPUT_FORMAT

+ ----------------

```
{ MLTOF -- Declaration of mlt$output_format }


TYPE
  mlt$output_format = record
    justification: mlt$justify,
    sign: mlt$sign_treatment,
    format: mlt$format,
    scale_factor: integer,
    width: mlt$string_length,
    digits: mlt$string_length,
    exponent_character: char,
    exponent_style: mlt$exponent_style,
  recend;
```

A1.17 MLT$SIGN_TREATMENT

\+ ----------------

```
{ MLTST -- Declaration of mlt$sign_treatment }


TYPE
  mlt$sign_treatment = (mlc$minus_if_negative, mlc$always_signed);
```

A1.18 MLT$STRING_LENGTH

\+ ----------------

```
{ MLTSL -- Declaration of mlt$string_length }


CONST
  mlc$min_string_length = 0,
  mlc$max_string_length = 7fffffff(16);

TYPE
  mlt$string_length = mlc$min_string_length ..
    mlc$max_string_length;
```

!

A1.19 MLT$VAX_FLOATING_TYPE

A1.19 MLT$VAX_FLOATING_TYPE

```
{ MLTVXFT -- Declaration of mlt$vax_floating_type }

TYPE
  mlt$vax_floating_type = (mlc$VAX_4_F_float, mlc$VAX_8_D_float,
        mlc$VAX_8_G_float, mlc$VAX_16_H_float);
```

A1.20 MLT$VAX_INTEGER_LENGTH

```
{ MLTVXIL -- Declaration of mlt$vax_integer_length }

CONST
  mlc$min_VAX_integer_length = 1,
  mlc$max_VAX_integer_length = 8;

TYPE
  mlt$VAX_integer_length = mlc$min_VAX_integer_length ..
        mlc$max_VAX_integer_length;
```

A1.21 MLT$VAX_LOGICAL_LENGTH

```
{ MLTVXLL -- Declaration of mlt$vax_logical_length }

TYPE
  mlt$vax_logical_length = (mlc$vax_logical_1, mlc$vax_logical_2,
        mlc$vax_logical_4);
```

A1.22 MLT$VAX_PACKED_DECIMAL_LENGTH

```
{ MLTVXDL -- Declaration of mlt$vax_packed_decimal_length }

TYPE
```

A1.22 MLT$VAX_PACKED_DECIMAL_LENGTH

```
mlt$vax_packed_decimal_length = 1 .. 19;
```


B1.0 CMML MATHEMATICAL ERRORS

+

The error numbers and message templates for the CMML Math Library functions are contained in this appendix. The function input parameter(s) are displayed along with each error message.

```
{ MLCBEN -- Definition of CMML base error number }


CONST
  mlc$base_err_num = 670000;



{ MLEACOS -- Error numbers for ACOS }


CONST

  mle$acos_arg_indef = mlc$base_err_num + 1,
  {F +N+P(+P). Argument indefinite.

  mle$acos_arg_inf = mlc$base_err_num + 2,
  {F +N+P(+P). Argument infinite.

  mle$acos_arg_range = mlc$base_err_num + 3
  {F +N+P(+P). Argument must be in range [-1.0,1.0].}

  ;



{ MLEAINT -- Error numbers for AINT }


CONST

  mle$aint_arg_indef = mlc$base_err_num + 4,
  {F +N+P(+P). Argument indefinite.}

  mle$aint_arg_inf = mlc$base_err_num + 5
  {F +N+P(+P). Argument infinite.}

  ;
```

```
{ MLEALN -- Error numbers for ALOG }

CONST

  mle$alog_arg_indef = mlc$base_err_num + 6,
  {F +N+P(+P). Argument indefinite.}

  mle$alog_arg_inf = mlc$base_err_num + 7,
  {F +N+P(+P). Argument infinite.}

  mle$alog_arg_0 = mlc$base_err_num + 8,
  {F +N+P(0.0). Argument must be > 0.0.}

  mle$alog_arg_neg = mlc$base_err_num + 9
  {F +N+P(+P). Argument must be > 0.0.}

  ;


{ MLEALOG -- Error numbers for ALOG10 }

CONST

  mle$alog10_arg_indef = mlc$base_err_num + 10,
  {F +N+P(+P). Argument indefinite.}

  mle$alog10_arg_inf = mlc$base_err_num + 11,
  {F +N+P(+P). Argument infinite.}

  mle$alog10_arg_0 = mlc$base_err_num + 12,
  {F +N+P(0.0). Argument must be > 0.0.}

  mle$alog10_arg_neg = mlc$base_err_num + 13
  {F +N+P(+P). Argument must be > 0.0.}

  ;


{ MLEAMOD -- Error numbers for AMOD }

CONST
```

```
  mle$amod_arg1_indef = mlc$base_err_num + 14,
  {F +N+P(arg1=+P,arg2=+P). Arg1 indefinite.}

  mle$amod_arg2_indef = mlc$base_err_num + 15,
  {F +N+P(arg1=+P,arg2=+P). Arg2 indefinite.}

  mle$amod_arg1_inf = mlc$base_err_num + 16,
  {F +N+P(arg1=+P,arg2=+P). Arg1 infinite.}

  mle$amod_arg2_inf = mlc$base_err_num + 17,
  {F +N+P(arg1=+P,arg2=+P). Arg2 infinite.}

  mle$amod_arg2_0 = mlc$base_err_num + 18,
  {F +N+P(arg1=+P,arg2=0.0). Arg2 must be nonzero.}

  mle$amod_args_range = mlc$base_err_num + 19
  {F +N+P(arg1=+P,arg2=+P). Arg1/arg2 infinite.}

  ;


{ MLEANIN -- Error numbers for ANINT }


CONST

  mle$anint_arg_indef = mlc$base_err_num + 20,
  {F +N+P(+P). Argument indefinite.}

  mle$anint_arg_inf = mlc$base_err_num + 21
  {F +N+P(+P). Arg infinite.}

  ;


{ MLEASIN -- Error numbers for ASIN }


CONST

  mle$asin_arg_indef = mlc$base_err_num + 22,
  {F +N+P(+P). Argument indefinite.}

  mle$asin_arg_inf = mlc$base_err_num + 23,
  {F +N+P(+P). Argument infinite.}
```

```
  mle$asin_arg_range = mlc$base_err_num + 24
  {F +N+P(+P). Argument must be in range [-1.0,1.0].}

  ;


{ MLEATAN -- Error numbers for ATAN }


CONST

  mle$atan_arg_indef = mlc$base_err_num + 25
  {F +N+P(+P). Argument indefinite.}

  ;


{ MLEATN2 -- Error numbers for ATAN2 }


CONST

  mle$atan2_arg1_indef = mlc$base_err_num + 26,
  {F +N+P(arg1=+P,arg2=+P). Arg1 indefinite.}

  mle$atan2_arg2_indef = mlc$base_err_num + 27,
  {F +N+P(arg1=+P,arg2=+P). Arg2 indefinite.}

  mle$atan2_args_inf = mlc$base_err_num + 28,
  {F +N+P(arg1=+P,arg2=+P). Both arguments infinite.}

  mle$atan2_args_0 = mlc$base_err_num + 29,
  {F +N+P(0.0,0.0). One argument must be nonzero.}

  mle$atan2_args_range = mlc$base_err_num + 30                          :
  {F +N+P(arg1=+P,arg2=+P). Arg2 must be zero if arg1/arg2              :
  {infinite.}
  ;


{ MLEATNH -- Error numbers for ATANH }


CONST

  mle$atanh_arg_indef = mlc$base_err_num + 31,
```



```
  {F +N+P(+P). Argument indefinite.}
```

```
  mle$atanh_arg_inf = mlc$base_err_num + 32,
  {F +N+P(+P). Argument infinite.}

  mle$atanh_arg_range = mlc$base_err_num + 33
  {F +N+P(+P). ABS(argument) must be < 1.0.}

  ;


{ MLECABS -- Error numbers for CABS }


CONST

  mle$cabs_arg_indef = mlc$base_err_num + 34,
  {F +N+P((+P,+P)). Argument indefinite.}

  mle$cabs_arg_inf = mlc$base_err_num + 35,
  {F +N+P((+P,+P)). Argument infinite.}

  mle$cabs_result_inf = mlc$base_err_num + 36
  {F +N+P((+P,+P)). Result infinite.}

  ;


{ MLECCOS -- Error numbers for CCOS }


CONST

  mle$ccos_arg_indef = mlc$base_err_num + 37,
  {F +N+P((+P,+P)). Argument indefinite.}

  mle$ccos_arg_inf = mlc$base_err_num + 38,
  {F +N+P((+P,+P)). Argument infinite.}

  mle$ccos_real_range = mlc$base_err_num + 39,
  {F +N+P((+P,+P)). ABS(real part) must be < 2.**47.}

  mle$ccos_imag_too_big = mlc$base_err_num + 40,
  {F +N+P((+P,+P)). Imag. part must be < 4095.*LOG(2).}

  mle$ccos_imag_too_small = mlc$base_err_num + 41
```



```
  {F +N+P((+P,+P)). Imag. part must be > -4095.*LOG(2).}
```

```
  ;


{ MLECEXP -- Error numbers for CEXP }

CONST

  mle$cexp_arg_indef = mlc$base_err_num + 42,
  {F +N+P((+P,+P)). Argument indefinite.}

  mle$cexp_arg_inf = mlc$base_err_num + 43,
  {F +N+P((+P,+P)). Argument infinite.}

  mle$cexp_imag_range = mlc$base_err_num + 44,
  {F +N+P((+P,+P)). ABS(imag. part) must be < 2.**47.}

  mle$cexp_real_range = mlc$base_err_num + 45
  {F +N+P((+P,+P)). ABS(real part) must be < 4095.*LOG(2).}

  ;


{ MLECLOG -- Error numbers for CLOG }

CONST

  mle$clog_arg_indef = mlc$base_err_num + 46,
  {F +N+P((+P,+P)). Argument indefinite.}

  mle$clog_arg_inf = mlc$base_err_num + 47,
  {F +N+P((+P,+P)). Argument infinite.}

  mle$clog_abs_arg_inf = mlc$base_err_num + 48,
  {F +N+P((+P,+P)). ABS(argument) infinite.}

  mle$clog_arg_0 = mlc$base_err_num + 49
  {F +N+P(0.0). One of real or imag. parts must be nonzero.}

  ;


{ MLECOS -- Error numbers for COS }
```



```
CONST
```

```
  mle$cos_arg_indef = mlc$base_err_num + 50,
  {F +N+P(+P). Argument indefinite.}

  mle$cos_arg_inf = mlc$base_err_num + 51,
  {F +N+P(+P). Argument infinite.}

  mle$cos_arg_range = mlc$base_err_num + 52
  {F +N+P(+P). ABS(argument) must be < 2.**47.}

  ;


{ MLECOSD -- Error numbers for COSD }


CONST

  mle$cosd_arg_indef = mlc$base_err_num + 247,
  {F +N+P(+P). Argument indefinite.}

  mle$cosd_arg_inf = mlc$base_err_num + 248,
  {F +N+P(+P). Argument infinite.}

  mle$cosd_arg_range = mlc$base_err_num + 249
  {F +N+P(+P). ABS(argument) must be < 2.**47.}

  ;


{ MLECOSH -- Error numbers for COSH }


CONST

  mle$cosh_arg_indef = mlc$base_err_num + 53,
  {F +N+P(+P). Argument indefinite.}

  mle$cosh_arg_inf = mlc$base_err_num + 54,
  {F +N+P(+P). Argument infinite.}

  mle$cosh_arg_range = mlc$base_err_num + 55
  {F +N+P(+P). ABS(argument) must be < 4095.*LOG(2).}
```



```
  ;
```

```
{ MLECOTAN -- Error numbers for COTAN }


CONST

  mle$cotan_arg_indef = mlc$base_err_num + 254,
  {F +N+P(+P). Argument indefinite.}

  mle$cotan_arg_inf = mlc$base_err_num + 255,
  {F +N+P(+P). Argument infinite.}

  mle$cotan_arg_range = mlc$base_err_num + 256,
  {F +N+P(+P). ABS(argument) must be < 2.**47.}

  mle$cotan_arg_0 = mlc$base_err_num + 265
  {F +N+P(0.0). Argument must be nonzero.}
  ;


{ MLECSIN -- Error numbers for CSIN }


CONST

  mle$csin_arg_indef = mlc$base_err_num + 56,
  {F +N+P((+P,+P)). Argument indefinite.}

  mle$csin_arg_inf = mlc$base_err_num + 57,
  {F +N+P((+P,+P)). Argument infinite.}

  mle$csin_real_range = mlc$base_err_num + 58,
  {F +N+P((+P,+P)). ABS(real part) must be < 2.**47.}

  mle$csin_imag_range = mlc$base_err_num + 59
  {F +N+P((+P,+P)). ABS(imag. part) must be < 4095.*LOG(2).}

  ;


{ MLECSQT -- Error numbers for CSQRT }
```



```
CONST

  mle$csqrt_arg_indef = mlc$base_err_num + 60,
  {F +N+P((+P,+P)). Argument indefinite.}
```

```
  mle$csqrt_arg_inf = mlc$base_err_num + 61,
  {F +N+P((+P,+P)). Argument infinite.}

  mle$csqrt_arg_range = mlc$base_err_num + 62
  {F +N+P((+P,+P)). ABS(argument) + ABS(real part) infinite.}

  ;


{ MLEDACS -- Error numbers for DACOS }


CONST

  mle$dacos_arg_indef = mlc$base_err_num + 63,
  {F +N+P(+P). Argument indefinite.}

  mle$dacos_arg_inf = mlc$base_err_num + 64,
  {F +N+P(+P). Argument infinite.}

  mle$dacos_arg_range = mlc$base_err_num + 65
  {F +N+P(+P). Argument must be in range [-1.0,1.0].}

  ;


{ MLEDASN -- Error numbers for DASIN }


CONST

  mle$dasin_arg_indef = mlc$base_err_num + 66,
  {F +N+P(+P). Argument indefinite.}

  mle$dasin_arg_inf = mlc$base_err_num + 67,
  {F +N+P(+P). Argument infinite.}

  mle$dasin_arg_range = mlc$base_err_num + 68
  {F +N+P(+P). Argument must be in range [-1.0,1.0].}

  ;
```



```
{ MLEDATN -- Error numbers for DATAN }


CONST
```

```
  mle$datan_arg_indef = mlc$base_err_num + 69
  {F +N+P(+P). Argument indefinite.}

  ;
```

```
{ MLEDTN2 -- Error numbers for DATAN2 }

CONST

  mle$datan2_arg1_indef = mlc$base_err_num + 70,
  {F +N+P(arg1=+P,arg2=+P). Arg1 indefinite.}

  mle$datan2_arg2_indef = mlc$base_err_num + 71,
  {F +N+P(arg1=+P,arg2=+P). Arg2 indefinite.}

  mle$datan2_args_inf = mlc$base_err_num + 72,
  {F +N+P(arg1=+P,arg2=+P). Arg1 and arg2 may not both be infinite.}

  mle$datan2_args_0 = mlc$base_err_num + 73
  {F +N+P(0.0,0.0). One of arg1 or arg2 must be nonzero.}

  ;
```

```
{ MLEDCOS -- Error numbers for DCOS }

CONST

  mle$dcos_arg_indef = mlc$base_err_num + 74,
  {F +N+P(+P). Argument indefinite.}

  mle$dcos_arg_inf = mlc$base_err_num + 75,
  {F +N+P(+P). Argument infinite.}

  mle$dcos_arg_range = mlc$base_err_num + 76
  {F +N+P(+P). ABS(argument) must be < 2.**47.}
```



```
  ;
```

```
{ MLEDCSH -- Error numbers for DCOSH }
```

```
CONST

  mle$dcosh_arg_indef = mlc$base_err_num + 77,
  {F +N+P(+P). Argument indefinite.}

  mle$dcosh_arg_inf = mlc$base_err_num + 78,
  {F +N+P(+P). Argument infinite.}

  mle$dcosh_arg_range = mlc$base_err_num + 79
  {F +N+P(+P). ABS(argument) must be < 4095.*LOG(2).}

  ;


{ MLEDDIM -- Error numbers for DDIM }


CONST

  mle$ddim_arg1_indef = mlc$base_err_num + 80,
  {F +N+P(arg1=+P,arg2=+P). Arg1 indefinite.}

  mle$ddim_arg2_indef = mlc$base_err_num + 81,
  {F +N+P(arg1=+P,arg2=+P). Arg2 indefinite.}

  mle$ddim_arg1_inf = mlc$base_err_num + 82,
  {F +N+P(arg1=+P,arg2=+P). Arg1 infinite.}

  mle$ddim_arg2_inf = mlc$base_err_num + 83,
  {F +N+P(arg1=+P,arg2=+P). Arg2 infinite.}

  mle$ddim_result_inf = mlc$base_err_num + 84
  {F +N+P(arg1=+P,arg2=+P). Result infinite.}

  ;


{ MLEDEXP -- Error numbers for DEXP }
```



```
CONST

  mle$dexp_arg_indef = mlc$base_err_num + 85,
  {F +N+P(+P). Argument indefinite.}

  mle$dexp_arg_inf = mlc$base_err_num + 86,
  {F +N+P(+P). Argument infinite.}
```

```
  mle$dexp_arg_too_big = mlc$base_err_num + 87,
  {F +N+P(+P). Argument must be < 4095.*LOG(2).}

  mle$dexp_arg_too_small = mlc$base_err_num + 88
  {F +N+P(+P). Argument must be > -4095.*LOG(2).}

  ;


{ MLEDIM -- Error numbers for DIM }


CONST

  mle$dim_arg1_indef = mlc$base_err_num + 89,
  {F +N+P(arg1=+P,arg2=+P). Arg1 indefinite.}

  mle$dim_arg2_indef = mlc$base_err_num + 90,
  {F +N+P(arg1=+P,arg2=+P). Arg2 indefinite.}

  mle$dim_arg1_inf = mlc$base_err_num + 91,
  {F +N+P(arg1=+P,arg2=+P). Arg1 infinite.}

  mle$dim_arg2_inf = mlc$base_err_num + 92,
  {F +N+P(arg1=+P,arg2=+P). Arg2 infinite.}

  mle$dim_result_inf = mlc$base_err_num + 93
  {F +N+P(arg1=+P,arg2=+P). Result infinite.}

  ;


{ MLEDINT -- Error numbers for DINT }


CONST

  mle$dint_arg_indef = mlc$base_err_num + 94,
  {F +N+P(+P). Argument indefinite.}
```



```
  mle$dint_arg_inf = mlc$base_err_num + 95
  {F +N+P(+P). Argument infinite.}

  ;


{ MLEDLN -- Error numbers for DLOG }
```

```
CONST

  mle$dlog_arg_indef = mlc$base_err_num + 96,
  {F +N+P(+P). Argument indefinite.}

  mle$dlog_arg_inf = mlc$base_err_num + 97,
  {F +N+P(+P). Argument infinite.}

  mle$dlog_arg_0 = mlc$base_err_num + 98,
  {F +N+P(0.0). Argument must be > 0.0.}

  mle$dlog_arg_neg = mlc$base_err_num + 99
  {F +N+P(+P). Argument must be > 0.0.}

  ;


{ MLEDLOG -- Error numbers for DLOG10 }


CONST

  mle$dlog10_arg_indef = mlc$base_err_num + 100,
  {F +N+P(+P). Argument indefinite.}

  mle$dlog10_arg_inf = mlc$base_err_num + 101,
  {F +N+P(+P). Argument infinite.}

  mle$dlog10_arg_0 = mlc$base_err_num + 102,
  {F +N+P(0.0). Argument must be > 0.0.}

  mle$dlog10_arg_neg = mlc$base_err_num + 103
  {F +N+P(+P). Argument must be > 0.0.}

  ;
```



```
{ MLEDMOD -- Error numbers for DMOD }


CONST

  mle$dmod_arg1_indef = mlc$base_err_num + 104,
  {F +N+P(arg1=+P,arg2=+P). Arg1 indefinite.}

  mle$dmod_arg2_indef = mlc$base_err_num + 105,
```

```
  {F +N+P(arg1=+P,arg2=+P). Arg2 indefinite.}

  mle$dmod_arg1_inf = mlc$base_err_num + 106,
  {F +N+P(arg1=+P,arg2=+P). Arg1 infinite.}

  mle$dmod_arg2_inf = mlc$base_err_num + 107,
  {F +N+P(arg1=+P,arg2=+P). Arg2 infinite.}

  mle$dmod_arg2_0 = mlc$base_err_num + 108,
  {F +N+P(arg1=+P,arg2=0.0). Arg2 must be nonzero.}

  mle$dmod_args_range = mlc$base_err_num + 109
  {F +N+P(arg1=+P,arg2=+P). Arg1/arg2 infinite.}

  ;


{ MLEDNIN -- Error numbers for DNINT }


CONST

  mle$dnint_arg_indef = mlc$base_err_num + 110,
  {F +N+P(+P). Argument indefinite.}

  mle$dnint_arg_inf = mlc$base_err_num + 111
  {F +N+P(+P). Argument infinite.}

  ;


{ MLEDPRD -- Error numbers for DPROD }


CONST

  mle$dprod_arg1_indef = mlc$base_err_num + 112,
```



```
  {F +N+P(arg1=+P,arg2=+P). Arg1 indefinite.}

  mle$dprod_arg2_indef = mlc$base_err_num + 113,
  {F +N+P(arg1=+P,arg2=+P). Arg2 indefinite.}

  mle$dprod_arg1_inf = mlc$base_err_num + 114,
  {F +N+P(arg1=+P,arg2=+P). Arg1 infinite.}

  mle$dprod_arg2_inf = mlc$base_err_num + 115,
  {F +N+P(arg1=+P,arg2=+P). Arg2 infinite.}
```

```
  mle$dprod_result_inf = mlc$base_err_num + 116
  {F +N+P(arg1=+P,arg2=+P). Result infinite.}

  ;


{ MLEDSIN -- Error numbers for DSIN }


CONST

  mle$dsin_arg_indef = mlc$base_err_num + 117,
  {F +N+P(+P). Argument indefinite.}

  mle$dsin_arg_inf = mlc$base_err_num + 118,
  {F +N+P(+P). Argument infinite.}

  mle$dsin_arg_range = mlc$base_err_num + 119
  {F +N+P(+P). ABS(argument) must be < 2.**47.}

  ;


{ MLEDSNH -- Error numbers for DSINH }


CONST

  mle$dsinh_arg_indef = mlc$base_err_num + 120,
  {F +N+P(+P). Argument indefinite.}

  mle$dsinh_arg_inf = mlc$base_err_num + 121,
  {F +N+P(+P). Argument infinite.}

  mle$dsinh_arg_range = mlc$base_err_num + 122
  {F +N+P(+P). ABS(argument) must be < 4095.*LOG(2).}
```



```
  ;


{ MLEDSQT -- Error numbers for DSQRT }


CONST

  mle$dsqrt_arg_indef = mlc$base_err_num + 123,
  {F +N+P(+P). Argument indefinite.}
```

```
  mle$dsqrt_arg_inf = mlc$base_err_num + 124,
  {F +N+P(+P). Argument infinite.}

  mle$dsqrt_arg_range = mlc$base_err_num + 125
  {F +N+P(+P). Argument must be >= 0.0.}

  ;


{ MLEDTAN -- Error numbers for DTAN }


CONST

  mle$dtan_arg_indef = mlc$base_err_num + 126,
  {F +N+P(+P). Argument indefinite.}

  mle$dtan_arg_inf = mlc$base_err_num + 127,
  {F +N+P(+P). Argument infinite.}

  mle$dtan_arg_range = mlc$base_err_num + 128
  {F +N+P(+P). ABS(argument) must be < 2.**47.}

  ;


{ MLEDTNH -- Error numbers for DTANH }


CONST

  mle$dtanh_arg_indef = mlc$base_err_num + 129
  {F +N+P(+P). Argument indefinite.}
```



```
  ;


{ MLEDTOD -- Error numbers for DTOD }


CONST

  mle$dtod_arg1_indef = mlc$base_err_num + 130,
  {F +N+P(arg1=+P,arg2=+P). Arg1 indefinite.}

  mle$dtod_arg2_indef = mlc$base_err_num + 131,
```

```
  {F +N+P(arg1=+P,arg2=+P). Arg2 indefinite.}

  mle$dtod_arg1_inf = mlc$base_err_num + 132,
  {F +N+P(arg1=+P,arg2=+P). Arg1 infinite.}

  mle$dtod_arg2_inf = mlc$base_err_num + 133,
  {F +N+P(arg1=+P,arg2=+P). Arg2 infinite.}

  mle$dtod_result_indef = mlc$base_err_num + 134,
  {F +N+P(arg1=0.0,arg2=+P). If arg1=0.0, arg2 must be > 0.0.}

  mle$dtod_arg1_neg = mlc$base_err_num + 135,
  {F +N+P(arg1=+P,arg2=+P). Arg1 must be >= 0.0.}

  mle$dtod_result_inf = mlc$base_err_num + 136
  {F +N+P(arg1=+P,arg2=+P). Result infinite.}

  ;


{ MLEDTOI -- Error numbers for DTOI }


CONST

  mle$dtoi_arg1_indef = mlc$base_err_num + 137,
  {F +N+P(arg1=+P,arg2=+P). Arg1 indefinite.}

  mle$dtoi_arg1_inf = mlc$base_err_num + 138,
  {F +N+P(arg1=+P,arg2=+P). Arg1 infinite.}

  mle$dtoi_result_indef = mlc$base_err_num + 139,
  {F +N+P(arg1=0.0,arg2=+P). If arg1=0.0, arg2 must be > 0.0.}

  mle$dtoi_result_inf = mlc$base_err_num + 140
```



```
  {F +N+P(arg1=+P,arg2=+P). Result infinite.}

  ;


{ MLEDTOX -- Error numbers for DTOX }


CONST

  mle$dtox_arg1_indef = mlc$base_err_num + 141,
  {F +N+P(arg1=+P,arg2=+P). Arg1 indefinite.}
```

```
  mle$dtox_arg2_indef = mlc$base_err_num + 142,
  {F +N+P(arg1=+P,arg2=+P). Arg2 indefinite.}

  mle$dtox_arg1_inf = mlc$base_err_num + 143,
  {F +N+P(arg1=+P,arg2=+P). Arg1 infinite.}

  mle$dtox_arg2_inf = mlc$base_err_num + 144,
  {F +N+P(arg1=+P,arg2=+P). Arg2 infinite.}

  mle$dtox_result_indef = mlc$base_err_num + 145,
  {F +N+P(arg1=0.0,arg2=+P). If arg1=0.0, arg2 must be > 0.0.}

  mle$dtox_arg1_neg = mlc$base_err_num + 146,
  {F +N+P(arg1=+P,arg2=+P). Arg1 must be >= 0.0.}

  mle$dtox_result_inf = mlc$base_err_num + 147
  {F +N+P(arg1=+P,arg2=+P). Result infinite.}

  ;


{ MLEDTOZ -- Error numbers for DTOZ }


CONST

  mle$dtoz_arg1_indef = mlc$base_err_num + 148,
  {F +N+P(+P,(+P,+P)). Arg1 indefinite.}

  mle$dtoz_arg2_indef = mlc$base_err_num + 149,
  {F +N+P(+P,(+P,+P)). Arg2 indefinite.}

  mle$dtoz_arg1_inf = mlc$base_err_num + 150,
  {F +N+P(+P,(+P,+P)). Arg1 infinite.}
```



```
  mle$dtoz_arg2_inf = mlc$base_err_num + 151,
  {F +N+P(+P,(+P,+P)). Arg2 infinite.}

  mle$dtoz_result_indef = mlc$base_err_num + 152,
  {F +N+P(0.0,(+P,+P)). Arg2 must be > 0.0.}

  mle$dtoz_arg1_neg = mlc$base_err_num + 153,
  {F +N+P(+P,(+P,+P)). Arg1 must be >= 0.0.}

  mle$dtoz_result_inf = mlc$base_err_num + 154
  {F +N+P(+P,(+P,+P)). Result infinite.}

  ;
```

```
{ MLEERF -- Error numbers for ERF }


CONST

  mle$erf_arg_indef = mlc$base_err_num + 155
  {F +N+P(+P). Argument indefinite.}

  ;


{ MLEERFC -- Error numbers for ERFC }


CONST

  mle$erfc_arg_indef = mlc$base_err_num + 156,
  {F +N+P(+P). Argument indefinite.}

  mle$erfc_arg_range = mlc$base_err_num + 184
  {F +N+P(+P). Argument must be <= 53.0374219959898.}

  ;


{ MLEEXP -- Error numbers for EXP }


CONST
```



```
  mle$exp_arg_indef = mlc$base_err_num + 157,
  {F +N+P(+P). Argument indefinite.}

  mle$exp_arg_inf = mlc$base_err_num + 158,
  {F +N+P(+P). Argument infinite.}

  mle$exp_arg_too_big = mlc$base_err_num + 159,
  {F +N+P(+P). Argument must be < 4095.*LOG(2).}

  mle$exp_arg_too_small = mlc$base_err_num + 160
  {F +N+P(+P). Argument must be > -4095.*LOG(2).}

  ;
```

```
{ MLEEXTB -- Error numbers for  EXTB }

  CONST

    mle$extb_arg1_neg = mlc$base_err_num + 257,
    {F +N+P(arg1=+P,arg2=+P). Starting bit must be >= 0.}

    mle$extb_arg2_neg = mlc$base_err_num + 258,
    {F +N+P(arg1=+P,arg2=+P). Length must be >= 0.}

    mle$extb_arg1_range = mlc$base_err_num + 259,
    {F +N+P(arg1=+P,arg2=+P). Starting bit must be < 64.}

    mle$extb_range = mlc$base_err_num + 260
    {F +N+P(arg1=+P,arg2=+P). Starting bit + Length must be <=64.}

    ;


  { MLEIDIM -- Error numbers for IDIM }


  CONST

    mle$idim_result_inf = mlc$base_err_num + 161
    {F +N+P(arg1=+P,arg2=+P). Arithmetic overflow.}

    ;
```



```
  { MLEIDNI -- Error numbers for IDNINT }


  CONST

    mle$idnint_arg_indef = mlc$base_err_num + 162,
    {F +N+P(+P). Argument indefinite.}

    mle$idnint_arg_inf = mlc$base_err_num + 163
    {F +N+P(+P). Argument infinite.}

    ;


{ MLEINSB -- Error numbers for  INSB }
```

```
CONST

  mle$insb_arg1_neg = mlc$base_err_num + 261,
  {F +N+P(arg1=+P,arg2=+P). Starting bit must be >= 0.}

  mle$insb_arg2_neg = mlc$base_err_num + 262,
  {F +N+P(arg1=+P,arg2=+P). Length must be >= 0.}

  mle$insb_arg1_range = mlc$base_err_num + 263,
  {F +N+P(arg1=+P,arg2=+P). Starting bit must be < 64.}

  mle$insb_range = mlc$base_err_num + 264
  {F +N+P(arg1=+P,arg2=+P). Starting bit + Length must be <=64.}

  ;


{ MLEITOD -- Error numbers for ITOD }


CONST

  mle$itod_arg2_indef = mlc$base_err_num + 164,
  {F +N+P(arg1=+P,arg2=+P). Arg2 indefinite.}

  mle$itod_arg2_inf = mlc$base_err_num + 165,
  {F +N+P(arg1=+P,arg2=+P). Arg2 infinite.}

  mle$itod_result_indef = mlc$base_err_num + 166,
  {F +N+P(arg1=0,arg2=+P). Arg2 must be > 0.0.}
```



```
  mle$itod_arg1_neg = mlc$base_err_num + 167,
  {F +N+P(arg1=+P,arg2=+P). Arg1 must be >= 0.0.}

  mle$itod_result_inf = mlc$base_err_num + 168
  {F +N+P(arg1=+P,arg2=+P). Result infinite.}

  ;


{ MLEITOI -- Error numbers for ITOI }


CONST

  mle$itoi_result_inf = mlc$base_err_num + 169,
  {F +N+P(arg1=+P,arg2=+P). Arithmetic overflow.}
```

```
  mle$itoi_result_indef = mlc$base_err_num + 170
  {F +N+P(arg1=0,arg2=+P). Arg2 must be > 0.0.}

  ;
```

```
{ MLEITOX -- Error numbers for ITOX }

CONST

  mle$itox_arg2_indef = mlc$base_err_num + 171,
  {F +N+P(arg1=+P,arg2=+P). Arg2 indefinite.}

  mle$itox_arg2_inf = mlc$base_err_num + 172,
  {F +N+P(arg1=+P,arg2=+P). Arg2 infinite.}

  mle$itox_result_indef = mlc$base_err_num + 173,
  {F +N+P(arg1=0,arg2=+P). Arg2 must be > 0.0.}

  mle$itox_arg1_neg = mlc$base_err_num + 174,
  {F +N+P(arg1=+P,arg2=+P). Arg1 must be >= 0.0.}

  mle$itox_result_inf = mlc$base_err_num + 175
  {F +N+P(arg1=+P,arg2=+P). Result infinite.}

  ;
```



```
{ MLEITOZ -- Error numbers for ITOZ }

CONST

  mle$itoz_arg2_indef = mlc$base_err_num + 176,
  {F +N+P(+P,(+P,+P)). Arg2 indefinite.}

  mle$itoz_arg2_inf = mlc$base_err_num + 177,
  {F +N+P(+P,(+P,+P)). Arg2 infinite.}

  mle$itoz_result_indef = mlc$base_err_num + 178,
  {F +N+P(0,(+P,+P)). Arg2 must be > 0.0.}

  mle$itoz_result_inf = mlc$base_err_num + 179,
  {F +N+P(+P,(+P,+P)). Result infinite.}

  mle$itoz_arg1_neg = mlc$base_err_num + 180
```

```
  {F +N+P(+P,(+P,+P)). Arg1 must be >= 0.0.}

  ;


{ MLEMOD -- Error numbers for MOD }


CONST

  mle$mod_arg2_0 = mlc$base_err_num + 181
  {F +N+P(arg1=+P,arg2=0). Arg2 must be nonzero.}

  ;


{ MLENINT -- Error numbers for NINT }


CONST

  mle$nint_arg_indef = mlc$base_err_num + 182,
  {F +N+P(+P). Argument indefinite.}

  mle$nint_arg_inf = mlc$base_err_num + 183
  {F +N+P(+P). Argument infinite.}

  ;
```



```
{ MLESIN -- Error numbers for SIN }


CONST

  mle$sin_arg_indef = mlc$base_err_num + 185,
  {F +N+P(+P). Argument indefinite.}

  mle$sin_arg_inf = mlc$base_err_num + 186,
  {F +N+P(+P). Argument infinite.}

  mle$sin_arg_range = mlc$base_err_num + 187
  {F +N+P(+P). ABS(argument) must be < 2.**47.}

  ;
```

```
{ MLESIND -- Error numbers for SIND }


CONST

  mle$sind_arg_indef = mlc$base_err_num + 244,
  {F +N+P(+P). Argument indefinite.}

  mle$sind_arg_inf = mlc$base_err_num + 245,
  {F +N+P(+P). Argument infinite.}

  mle$sind_arg_range = mlc$base_err_num + 246
  {F +N+P(+P). ABS(argument) must be < 2.**47.}

  ;



{ MLESINH -- Error numbers for SINH }


CONST

  mle$sinh_arg_indef = mlc$base_err_num + 188,
  {F +N+P(+P). Argument indefinite.}

  mle$sinh_arg_inf = mlc$base_err_num + 189,
  {F +N+P(+P). Argument infinite.}
```



```
  mle$sinh_arg_range = mlc$base_err_num + 190
  {F +N+P(+P). ABS(argument) must be < 4095.*LOG(2).}

  ;



{ MLESQRT -- Error numbers for SQRT }


CONST

  mle$sqrt_arg_indef = mlc$base_err_num + 191,
  {F +N+P(+P). Argument indefinite.}

  mle$sqrt_arg_inf = mlc$base_err_num + 192,
  {F +N+P(+P). Argument infinite.}

  mle$sqrt_arg_neg = mlc$base_err_num + 193
  {F +N+P(+P). Argument must be >= 0.0.}
```

```
  ;


{ MLETAN -- Error numbers for TAN }


CONST

  mle$tan_arg_indef = mlc$base_err_num + 194,
  {F +N+P(+P). Argument indefinite.}

  mle$tan_arg_inf = mlc$base_err_num + 195,
  {F +N+P(+P). Argument infinite.}

  mle$tan_arg_range = mlc$base_err_num + 196
  {F +N+P(+P). ABS(argument) must be < 2.**47.}

  ;


{ MLETAND -- Error numbers for TAND }


CONST
```



```
  mle$tand_arg_indef = mlc$base_err_num + 250,
  {F +N+P(+P). Argument indefinite.}

  mle$tand_arg_inf = mlc$base_err_num + 251,
  {F +N+P(+P). Argument infinite.}

  mle$tand_arg_range = mlc$base_err_num + 252,
  {F +N+P(+P). ABS(argument) must be < 2.**47.}

  mle$tand_result_inf = mlc$base_err_num + 253
  {F +N+P(+P). Argument must not be an exact odd multiple of 90.0.}

  ;


{ MLETANH -- Error numbers for TANH }


CONST

  mle$tanh_arg_indef = mlc$base_err_num + 197
```

```
  {F +N+P(+P). Argument indefinite.}

  ;


{ MLEXTOD -- Error numbers for XTOD }


CONST

  mle$xtod_arg1_indef = mlc$base_err_num + 198,
  {F +N+P(arg1=+P,arg2=+P). Arg1 indefinite.}

  mle$xtod_arg2_indef = mlc$base_err_num + 199,
  {F +N+P(arg1=+P,arg2=+P). Arg2 indefinite.}

  mle$xtod_arg1_inf = mlc$base_err_num + 200,
  {F +N+P(arg1=+P,arg2=+P). Arg1 infinite.}

  mle$xtod_arg2_inf = mlc$base_err_num + 201,
  {F +N+P(arg1=+P,arg2=+P). Arg2 infinite.}

  mle$xtod_result_indef = mlc$base_err_num + 202,                    !
  {F +N+P(arg1=0.0,arg2=+P). Arg2 must be >= 0.0.}                   !

  mle$xtod_arg1_neg = mlc$base_err_num + 203,
```



```
  {F +N+P(arg1=+P,arg2=+P). Arg1 must be >= 0.0.}

  mle$xtod_result_inf = mlc$base_err_num + 204
  {F +N+P(arg1=+P,arg2=+P). Result infinite.}

  ;


{ MLEXTOI -- Error numbers for XTOI }


CONST

  mle$xtoi_arg1_indef = mlc$base_err_num + 205,
  {F +N+P(arg1=+P,arg2=+P). Arg1 indefinite.}

  mle$xtoi_arg1_inf = mlc$base_err_num + 206,
  {F +N+P(arg1=+P,arg2=+P). Arg1 infinite.}

  mle$xtoi_result_indef = mlc$base_err_num + 207,                    !
  {F +N+P(arg1=0.0,arg2=+P). Arg2 must be >= 0.}                     !
```

```
  mle$xtoi_result_inf = mlc$base_err_num + 208
  {F +N+P(arg1=+P,arg2=+P). Result infinite.}

  ;


{ MLEXTOX -- Error numbers for XTOX }


CONST

  mle$xtox_arg1_indef = mlc$base_err_num + 209,
  {F +N+P(arg1=+P,arg2=+P). Arg1 indefinite.}

  mle$xtox_arg2_indef = mlc$base_err_num + 210,
  {F +N+P(arg1=+P,arg2=+P). Arg2 indefinite.}

  mle$xtox_arg1_inf = mlc$base_err_num + 211,
  {F +N+P(arg1=+P,arg2=+P). Arg1 infinite.}

  mle$xtox_arg2_inf = mlc$base_err_num + 212,
  {F +N+P(arg1=+P,arg2=+P). Arg2 infinite.}

  mle$xtox_result_indef = mlc$base_err_num + 213,
  {F +N+P(arg1=0.0,arg2=+P). Arg2 must be > 0.0.}
```



```
  mle$xtox_arg1_neg = mlc$base_err_num + 214,
  {F +N+P(arg1=+P,arg2=+P). Arg1 must be >= 0.0.}

  mle$xtox_result_inf = mlc$base_err_num + 215
  {F +N+P(arg1=+P,arg2=+P). Result infinite.}

  ;


{ MLEXTOZ -- Error numbers for XTOZ }


CONST

  mle$xtoz_arg1_indef = mlc$base_err_num + 216,
  {F +N+P(+P,(+P,+P)). Arg1 indefinite.}

  mle$xtoz_arg2_indef = mlc$base_err_num + 217,
  {F +N+P(+P,(+P,+P)). Arg2 indefinite.}

  mle$xtoz_arg1_inf = mlc$base_err_num + 218,
  {F +N+P(+P,(+P,+P)). Arg1 infinite.}
```

```
  mle$xtoz_arg2_inf = mlc$base_err_num + 219,
  {F +N+P(+P,(+P,+P)). Arg2 infinite.}

  mle$xtoz_result_indef = mlc$base_err_num + 220,
  {F +N+P(0.0,(+P,+P)). Arg2 must be > 0.0.}

  mle$xtoz_result_inf = mlc$base_err_num + 221
  {F +N+P(+P,(+P,+P)). Result infinite.}

  ;


{ MLEZTOD -- Error numbers for ZTOD }


CONST

  mle$ztod_arg1_indef = mlc$base_err_num + 222,
  {F +N+P((+P,+P),+P). Arg1 indefinite.}

  mle$ztod_arg2_indef = mlc$base_err_num + 223,
  {F +N+P((+P,+P),+P). Arg2 indefinite.}
```



```
  mle$ztod_arg1_inf = mlc$base_err_num + 224,
  {F +N+P((+P,+P),+P). Arg1 infinite.}

  mle$ztod_arg2_inf = mlc$base_err_num + 225,
  {F +N+P((+P,+P),+P). Arg2 infinite.}

  mle$ztod_result_indef = mlc$base_err_num + 226,
  {F +N+P(0.0,+P). Arg2 must be > 0.0.}

  mle$ztod_result_inf = mlc$base_err_num + 227
  {F +N+P((+P,+P),+P). Result infinite.}

  ;


{ MLEZTOI -- Error numbers for ZTOI }


CONST

  mle$ztoi_arg1_indef = mlc$base_err_num + 228,
  {F +N+P((+P,+P),+P). Arg1 indefinite.}

  mle$ztoi_arg1_inf = mlc$base_err_num + 229,
```

```
  {F +N+P((+P,+P),+P). Arg1 infinite.}

  mle$ztoi_result_inf = mlc$base_err_num + 230,
  {F +N+P((+P,+P),+P). Result infinite.}

  mle$ztoi_result_indef = mlc$base_err_num + 231                         :
  {F +N+P(0.0,+P). Arg2 must be > 0.0.}                                  :

  ;


{ MLEZTOX -- Error numbers for ZTOX }


CONST

  mle$ztox_arg1_indef = mlc$base_err_num + 232,
  {F +N+P((+P,+P),+P). Arg1 indefinite.}

  mle$ztox_arg2_indef = mlc$base_err_num + 233,
  {F +N+P((+P,+P),+P). Arg2 indefinite.}

  mle$ztox_arg1_inf = mlc$base_err_num + 234,
```



```
  {F +N+P((+P,+P),+P). Arg1 infinite.}

  mle$ztox_arg2_inf = mlc$base_err_num + 235,                           :
  {F +N+P((+P,+P),+P). Arg2 must be > 0.0.}                             :

  mle$ztox_result_indef = mlc$base_err_num + 236,                       :
  {F +N+P(0.0,+P). Arg2 must be > 0.0.}                                 :

  mle$ztox_result_inf = mlc$base_err_num + 237
  {F +N+P((+P,+P),+P). Result infinite.}

  ;


{ MLEZTOZ -- Error numbers for ZTOZ }


CONST

  mle$ztoz_arg1_indef = mlc$base_err_num + 238,
  {F +N+P((+P,+P),(+P,+P)). Arg1 indefinite.}

  mle$ztoz_arg2_indef = mlc$base_err_num + 239,
  {F +N+P((+P,+P),(+P,+P)). Arg2 indefinite.}
```

```
mle$ztoz_arg1_inf = mlc$base_err_num + 240,
{F +N+P((+P,+P),(+P,+P). Arg1 infinite.}

mle$ztoz_arg2_inf = mlc$base_err_num + 241,
{F +N+P((+P,+P),(+P,+P)). Arg2 infinite.}

mle$ztoz_result_indef = mlc$base_err_num + 242,                      !
{F +N+P(0.0,(+P,+P)). Arg1 must be nonzero.}                          !

mle$ztoz_result_inf = mlc$base_err_num + 243
{F +N+P((+P,+P),(+P,+P)). Result infinite.}

;
```


C1.0 MADIFY TO SCU CONVERSION

+

The following is a listing of the file used to convert the CMML common deck PL from MADIFY to SCU format.

```
OLD_NAME=MLCBEN  NN=MLC$BASE_ERR_NUM             MN=MLCBEN       !
OLD_NAME=MLD$78B NN=MLP$170_TO_180_BINARY        MN=MLD$78B      !
OLD_NAME=MLD$78F NN=MLP$170_TO_180_FLOATING      MN=MLD$78F      !
OLD_NAME=MLD$78I NN=MLP$170_TO_180_INTEGER       MN=MLD$78I      !
OLD_NAME=MLD$87B NN=MLP$180_TO_170_BINARY        MN=MLD$87B      !
OLD_NAME=MLD$87F NN=MLP$180_TO_170_FLOATING      MN=MLD$87F      !
OLD_NAME=MLD$87I NN=MLP$180_TO_170_INTEGER       MN=MLD$87I      !
OLD_NAME=MLD$BDP NN=MLP$BDP_CONVERSION           MN=MLD$BDP      !
OLD_NAME=MLD$BIT NN=MLP$BITS_TO_AND_FROM_BDP     MN=MLD$BIT      !
OLD_NAME=MLD$CCI NN=MLP$COMPARE_COLLATED         MN=MLD$CCI      !
OLD_NAME=MLD$CF  NN=MLP$COMPARE_FLOATING         MN=MLD$CF       !
OLD_NAME=MLD$CFI NN=MLP$CONVERT_FLOAT_TO_INTEGE  MN=MLD$CFI      !
OLD_NAME=MLD$CFN NN=MLP$COMPUTE_FLOATING_NUMBER  MN=MLD$CFN      !
OLD_NAME=MLD$CIF NN=MLP$CONVERT_INTEGER_TO_FLOAT MN=MLD$CIF      !
OLD_NAME=MLD$CMN NN=MLP$COMPARE_BDP              MN=MLD$CMN      !
OLD_NAME=MLD$COM NN=MLP$COMPARE_BYTES            MN=MLD$COM      !
OLD_NAME=MLD$IBN NN=MLP$INPUT_BASE_NUMBER        MN=MLD$IBN      !
OLD_NAME=MLD$IFM NN=MLP$INPUT_FLOATING_MANTISSA  MN=MLD$IFM      !
OLD_NAME=MLD$IFN NN=MLP$INPUT_FLOATING_NUMBER    MN=MLD$IFN      !
```

```
OLD_NAME=MLD$II  NN=MLP$INPUT_INTEGER          MN=MLD$II          !
OLD_NAME=MLD$IUD NN=MLP$INPUT_UNPACKED_DECIMAL MN=MLD$IUD         !
OLD_NAME=MLD$MOV NN=MLP$MOVE_BYTES             MN=MLD$MOV         !
OLD_NAME=MLD$OBN NN=MLP$OUTPUT_BASE_NUMBER     MN=MLD$OBN         !
OLD_NAME=MLD$OFD NN=MLP$OUTPUT_FLOATING_DIGITS MN=MLD$OFD         !
OLD_NAME=MLD$OFN NN=MLP$OUTPUT_FLOATING_NUMBER MN=MLD$OFN         !
OLD_NAME=MLD$OI  NN=MLP$OUTPUT_INTEGER         MN=MLD$OI          !
OLD_NAME=MLD$RFN NN=MLP$ROUND_FLOATING_NUMBER  MN=MLD$RFN         !
OLD_NAME=MLD$SCA NN=MLP$SCAN_BYTES             MN=MLD$SCA         !
OLD_NAME=MLD$SFN NN=MLP$SCALE_FLOATING_NUMBER  MN=MLD$SFN         !
OLD_NAME=MLD$TEX NN=MLP$TEST_FOR_EXCEPTION     MN=MLD$TEX         !
OLD_NAME=MLD$TRA NN=MLP$TRANSLATE_BYTES        MN=MLD$TRA         !
OLD_NAME=MLD$TYP NN=MLT$ALL_CMML_TYPES         MN=MLD$TYP         !
OLD_NAME=MLDECC  NN=MLE$EXCEPTION_CONDITION_CODES   MN=MLDECC     !
OLD_NAME=MLEACOS NN=MLE$ACOS                   MN=MLEACOS         !
OLD_NAME=MLEAINT NN=MLE$AINT                   MN=MLEAINT         !
OLD_NAME=MLEALN  NN=MLE$ALOG                   MN=MLEALN          !
OLD_NAME=MLEALOG NN=MLE$ALOG10                 MN=MLEALOG         !
OLD_NAME=MLEAMOD NN=MLE$AMOD                   MN=MLEAMOD         !
OLD_NAME=MLEANIN NN=MLE$ANINT                  MN=MLEANIN         !
OLD_NAME=MLEASIN NN=MLE$ASIN                   MN=MLEASIN         !
OLD_NAME=MLEATAN NN=MLE$ATAN                   MN=MLEATAN         !
OLD_NAME=MLEATN2 NN=MLE$ATAN2                  MN=MLEATN2         !
```



```
OLD_NAME=MLEATNH NN=MLE$ATANH                  MN=MLEATNH         !
OLD_NAME=MLECABS NN=MLE$CABS                   MN=MLECABS         !
OLD_NAME=MLECCOS NN=MLE$CCOS                   MN=MLECCOS         !
OLD_NAME=MLECEXP NN=MLE$CEXP                   MN=MLECEXP         !
OLD_NAME=MLECLOG NN=MLE$CLOG                   MN=MLECLOG         !
OLD_NAME=MLECOS  NN=MLE$COS                    MN=MLECOS          !
OLD_NAME=MLECOSD NN=MLE$COSD                   MN=MLECOSD         !
OLD_NAME=MLECOSH NN=MLE$COSH                   MN=MLECOSH         !
OLD_NAME=MLECSIN NN=MLE$CSIN                   MN=MLECSIN         !
OLD_NAME=MLECSQT NN=MLE$CSQRT                  MN=MLECSQT         !
OLD_NAME=MLEDACS NN=MLE$DACOS                  MN=MLEDACS         !
OLD_NAME=MLEDASN NN=MLE$DASIN                  MN=MLEDASN         !
OLD_NAME=MLEDATN NN=MLE$DATN                   MN=MLEDATN         !
OLD_NAME=MLEDCOS NN=MLE$DCOS                   MN=MLEDCOS         !
OLD_NAME=MLEDCSH NN=MLE$DCOSH                  MN=MLEDCSH         !
OLD_NAME=MLEDDIM NN=MLE$DDIM                   MN=MLEDDIM         !
OLD_NAME=MLEDEXP NN=MLE$DEXP                   MN=MLEDEXP         !
OLD_NAME=MLEDIM  NN=MLE$DIM                    MN=MLEDIM          !
OLD_NAME=MLEDINT NN=MLE$DINT                   MN=MLEDINT         !
OLD_NAME=MLEDLN  NN=MLE$DLOG                   MN=MLEDLN          !
OLD_NAME=MLEDLOG NN=MLE$DLOG10                 MN=MLEDLOG         !
OLD_NAME=MLEDMOD NN=MLE$DMOD                   MN=MLEDMOD         !
OLD_NAME=MLEDNIN NN=MLE$DNINT                  MN=MLEDNIN         !
OLD_NAME=MLEDPRD NN=MLE$DPROD                  MN=MLEDPRD         !
OLD_NAME=MLEDSIN NN=MLE$DSIN                   MN=MLEDSIN         !
OLD_NAME=MLEDSNH NN=MLE$DSINH                  MN=MLEDSNH         !
OLD_NAME=MLEDSQT NN=MLE$DSQRT                  MN=MLEDSQT         !
```

```
OLD_NAME=MLEDTAN NN=MLE$DTAN                  MN=MLEDTAN        !
OLD_NAME=MLEDTN2 NN=MLE$DATAN2                MN=MLEDTN2        !
OLD_NAME=MLEDTNH NN=MLE$DTANH                 MN=MLEDTNH        !
OLD_NAME=MLEDTOD NN=MLE$DTOD                  MN=MLEDTOD        !
OLD_NAME=MLEDTOI NN=MLE$DTOI                  MN=MLEDTOI        !
OLD_NAME=MLEDTOX NN=MLE$DTOX                  MN=MLEDTOX        !
OLD_NAME=MLEDTOZ NN=MLE$DTOZ                  MN=MLEDTOZ        !
OLD_NAME=MLEERF  NN=MLE$ERF                   MN=MLEERF         !
OLD_NAME=MLEERFC NN=MLE$ERFC                  MN=MLEERFC        !
OLD_NAME=MLEEXP  NN=MLE$EXP                   MN=MLEEXP         !
OLD_NAME=MLEIDIM NN=MLE$IDIM                  MN=MLEIDIM        !
OLD_NAME=MLEIDNI NN=MLE$IDNINT                MN=MLEIDNI        !
OLD_NAME=MLEITOD NN=MLE$ITOD                  MN=MLEITOD        !
OLD_NAME=MLEITOI NN=MLE$ITOI                  MN=MLEITOI        !
OLD_NAME=MLEITOX NN=MLE$ITOX                  MN=MLEITOX        !
OLD_NAME=MLEITOZ NN=MLE$ITOZ                  MN=MLEITOZ        !
OLD_NAME=MLEMOD  NN=MLE$MOD                   MN=MLEMOD         !
OLD_NAME=MLENINT NN=MLE$NINT                  MN=MLENINT        !
OLD_NAME=MLESIN  NN=MLE$SIN                   MN=MLESIN         !
OLD_NAME=MLESIND NN=MLE$SIND                  MN=MLESIND        !
OLD_NAME=MLESINH NN=MLE$SINH                  MN=MLESINH        !
OLD_NAME=MLESQRT NN=MLE$SQRT                   MN=MLESQRT       !
```



```
OLD_NAME=MLETAN  NN=MLE$TAN                   MN=MLETAN         !
OLD_NAME=MLETAND NN=MLE$TAND                  MN=MLETAND        !
OLD_NAME=MLETANH NN=MLE$TANH                  MN=MLETANH        !
OLD_NAME=MLEXTOD NN=MLE$XTOD                  MN=MLEXTOD        !
OLD_NAME=MLEXTOI NN=MLE$XTOI                  MN=MLEXTOI        !
OLD_NAME=MLEXTOX NN=MLE$XTOX                  MN=MLEXTOX        !
OLD_NAME=MLEXTOZ NN=MLE$XTOZ                  MN=MLEXTOZ        !
OLD_NAME=MLEZTOD NN=MLE$ZTOD                  MN=MLEZTOD        !
OLD_NAME=MLEZTOI NN=MLE$ZTOI                  MN=MLEZTOI        !
OLD_NAME=MLEZTOX NN=MLE$ZTOX                  MN=MLEZTOX        !
OLD_NAME=MLEZTOZ NN=MLE$ZTOZ                  MN=MLEZTOZ        !
OLD_NAME=MLPABS  NN=MLP$RABS                  MN=MLPABS         !
OLD_NAME=MLPACOS NN=MLP$RACOS                 MN=MLPACOS        !
OLD_NAME=MLPAIMG NN=MLP$RAIMAG                MN=MLPAIMG        !
OLD_NAME=MLPAINT NN=MLP$RAINT                 MN=MLPAINT        !
OLD_NAME=MLPALOG NN=MLP$RALOG                 MN=MLPALOG        !
OLD_NAME=MLPAMOD NN=MLP$RAMOD                 MN=MLPAMOD        !
OLD_NAME=MLPASIN NN=MLP$RASIN                 MN=MLPASIN        !
OLD_NAME=MLPATAN NN=MLP$RATAN                 MN=MLPATAN        !
OLD_NAME=MLPATN2 NN=MLP$RATAN2                MN=MLPATN2        !
OLD_NAME=MLPATNH NN=MLP$RATANH                MN=MLPATNH        !
OLD_NAME=MLPCABS NN=MLP$RCABS                 MN=MLPCABS        !
OLD_NAME=MLPCCOS NN=MLP$RCCOS                 MN=MLPCCOS        !
OLD_NAME=MLPCEXP NN=MLP$RCEXP                 MN=MLPCEXP        !
OLD_NAME=MLPCLOG NN=MLP$RCLOG                 MN=MLPCLOG        !
OLD_NAME=MLPCNJG NN=MLP$RCONJG                MN=MLPCNJG        !
OLD_NAME=MLPCOS  NN=MLP$RCOS                  MN=MLPCOS         !
OLD_NAME=MLPCOSD NN=MLP$RCOSD                 MN=MLPCOSD        !
```

```
OLD_NAME=MLPCOSH NN=MLP$RCOSH                 MN=MLPCOSH                   ..
OLD_NAME=MLPCSIN NN=MLP$RCSIN                 MN=MLPCSIN                   ..
OLD_NAME=MLPCSQT NN=MLP$RCSQRT                MN=MLPCSQT                   ..
OLD_NAME=MLPDABS NN=MLP$RDABS                 MN=MLPDABS                   ..
OLD_NAME=MLPDACS NN=MLP$RDACOS                MN=MLPDACS                   ..
OLD_NAME=MLPDASN NN=MLP$RDASIN                MN=MLPDASN                   ..
OLD_NAME=MLPDATN NN=MLP$RDATAN                MN=MLPDATN                   ..
OLD_NAME=MLPDCOS NN=MLP$RDCOS                 MN=MLPDCOS                   ..
OLD_NAME=MLPDCSH NN=MLP$RDCOSH                MN=MLPDCSH                   ..
OLD_NAME=MLPDDIM NN=MLP$RDDIM                 MN=MLPDDIM                   ..
OLD_NAME=MLPDEXP NN=MLP$RDEXP                 MN=MLPDEXP                   ..
OLD_NAME=MLPDIM  NN=MLP$RDIM                  MN=MLPDIM                    ..
OLD_NAME=MLPDINT NN=MLP$RDINT                 MN=MLPDINT                   ..
OLD_NAME=MLPDL10 NN=MLP$RDLOG10               MN=MLPDL10                   ..
OLD_NAME=MLPDLOG NN=MLP$RDLOG                 MN=MLPDLOG                   ..
OLD_NAME=MLPDMOD NN=MLP$RDMOD                 MN=MLPDMOD                   ..
OLD_NAME=MLPDNIT NN=MLP$RDNINT                MN=MLPDNIT                   ..
OLD_NAME=MLPDPRD NN=MLP$DPROD                 MN=MLPDPRD                   ..
OLD_NAME=MLPDSGN NN=MLP$RDSIGN                MN=MLPDSGN                   ..
OLD_NAME=MLPDSIN NN=MLP$RDSIN                 MN=MLPDSIN                   ..
OLD_NAME=MLPDSNH NN=MLP$RDSINH                MN=MLPDSNH                   ..
```



```
OLD_NAME=MLPDSQT NN=MLP$RDSQRT                MN=MLPDSQT                   ..
OLD_NAME=MLPDTAN NN=MLP$RDTAN                 MN=MLPDTAN                   ..
OLD_NAME=MLPDTN2 NN=MLP$RDATAN2               MN=MLPDTN2                   ..
OLD_NAME=MLPDTNH NN=MLP$RDTANH                MN=MLPDTNH                   ..
OLD_NAME=MLPDTOD NN=MLP$RDTOD                 MN=MLPDTOD                   ..
OLD_NAME=MLPDTOI NN=MLP$RDTOI                 MN=MLPDTOI                   ..
OLD_NAME=MLPDTOX NN=MLP$RDTOX                 MN=MLPDTOX                   ..
OLD_NAME=MLPDTOZ NN=MLP$RDTOZ                 MN=MLPDTOZ                   ..
OLD_NAME=MLPERF  NN=MLP$RERF                  MN=MLPERF                    ..
OLD_NAME=MLPERFC NN=MLP$RERFC                 MN=MLPERFC                   ..
OLD_NAME=MLPEXP  NN=MLP$REXP                  MN=MLPEXP                    ..
OLD_NAME=MLPIABS NN=MLP$RIABS                 MN=MLPIABS                   ..
OLD_NAME=MLPIDIM NN=MLP$RIDIM                 MN=MLPIDIM                   ..
OLD_NAME=MLPIDNT NN=MLP$RIDNINT               MN=MLPIDNT                   ..
OLD_NAME=MLPISGN NN=MLP$RISIGN                MN=MLPISGN                   ..
OLD_NAME=MLPITOD NN=MLP$RITOD                 MN=MLPITOD                   ..
OLD_NAME=MLPITOI NN=MLP$RITOI                 MN=MLPITOI                   ..
OLD_NAME=MLPITOX NN=MLP$RITOX                 MN=MLPITOX                   ..
OLD_NAME=MLPITOZ NN=MLP$RITOZ                 MN=MLPITOZ                   ..
OLD_NAME=MLPLG10 NN=MLP$RALOG10               MN=MLPLG10                   ..
OLD_NAME=MLPMOD  NN=MLP$RMOD                  MN=MLPMOD                    ..
OLD_NAME=MLPNIT  NN=MLP$RNINT                 MN=MLPNIT                    ..
OLD_NAME=MLPRANF NN=MLP$RRANF                 MN=MLPRANF                   ..
OLD_NAME=MLPSIGN NN=MLP$RSIGN                 MN=MLPSIGN                   ..
OLD_NAME=MLPSIN  NN=MLP$RSIN                  MN=MLPSIN                    ..
OLD_NAME=MLPSIND NN=MLP$RSIND                 MN=MLPSIND                   ..
OLD_NAME=MLPSINH NN=MLP$RSINH                 MN=MLPSINH                   ..
OLD_NAME=MLPSQRT NN=MLP$RSQRT                 MN=MLPSQRT                   ..
OLD_NAME=MLPTAN  NN=MLP$RTAN                  MN=MLPTAN                    ..
```

```
OLD_NAME=MLPTAND NN=MLP$RTAND                    MN=MLPTAND              !
OLD_NAME=MLPTANH NN=MLP$RTANH                    MN=MLPTANH              !
OLD_NAME=MLPXTOD NN=MLP$RXTOD                    MN=MLPXTOD              !
OLD_NAME=MLPXTOI NN=MLP$RXTOI                    MN=MLPXTOI              !
OLD_NAME=MLPXTOX NN=MLP$RXTOX                    MN=MLPXTOX              !
OLD_NAME=MLPXTOZ NN=MLP$RXTOZ                    MN=MLPXTOZ              !
OLD_NAME=MLPZTOD NN=MLP$RZTOD                    MN=MLPZTOD              !
OLD_NAME=MLPZTOI NN=MLP$RZTOI                    MN=MLPZTOI              !
OLD_NAME=MLPZTOX NN=MLP$RZTOX                    MN=MLPZTOX              !
OLD_NAME=MLPZTOZ NN=MLP$RZTOZ                    MN=MLPZTOZ              !
OLD_NAME=MLTBDP  NN=MLT$BDP_TYPE                 MN=MLTBDP               !
OLD_NAME=MLTBDPL NN=MLT$BDP_LENGTH               MN=MLTBDPL              !
OLD_NAME=MLTC    NN=MLT$COMPLEX                  MN=MLTC                 !
OLD_NAME=MLTCOMP NN=MLT$COMPARE                  MN=MLTCOMP              !
OLD_NAME=MLTDSL  NN=MLT$DIGIT_STRING_LENGTH      MN=MLTDSL               !
OLD_NAME=MLTERR  NN=MLT$ERROR                    MN=MLTERR               !
OLD_NAME=MLTES   NN=MLT$EXPONENT_STYLE           MN=MLTES                !
OLD_NAME=MLTFI   NN=MLT$FLOATING_INPUT           MN=MLTFI                !
OLD_NAME=MLTFL   NN=MLT$FLOATING_LENGTH          MN=MLTFL                !
OLD_NAME=MLTFORM NN=MLT$FORMAT                   MN=MLTFORM              !
```



```
OLD_NAME=MLTHB   NN=MLT$HANDLE_BLANKS            MN=MLTHB                !
OLD_NAME=MLTIL   NN=MLT$INTEGER_LENGTH           MN=MLTIL                !
OLD_NAME=MLTIT   NN=MLT$INTEGER_TYPE             MN=MLTIT                !
OLD_NAME=MLTJUST NN=MLT$JUSTIFY                  MN=MLTJUST              !
OLD_NAME=MLTLR   NN=MLT$LONGREAL                 MN=MLTLR                !
OLD_NAME=MLTNDB  NN=MLT$NON_DECIMAL_BASE         MN=MLTNDB               !
OLD_NAME=MLTOF   NN=MLT$OUTPUT_FORMAT            MN=MLTOF                !
OLD_NAME=MLTSL   NN=MLT$STRING_LENGTH            MN=MLTSL                !
OLD_NAME=MLTST   NN=MLT$SIGN_TREATMENT           MN=MLTST                !
OLD_NAME=MLVSTAT NN=MLV$STAT                     MN=MLVSTAT              !
```

Table of Contents

1.0 PREFACE 1-1
1.1 PREFACE TO REVISION E 1-1
1.2 SCOPE 1-1
1.3 REFERENCES 1-2

2.0 MATHEMATICAL FUNCTIONS AND ROUTINES 2-1
2.1 INTRODUCTION 2-1
2.2 NUMBER TYPES 2-1
2.3 GENERAL RULES 2-3
2.4 DOCUMENTATION CONVENTIONS 2-3
2.5 LINKAGE INTERFACE 2-4
2.6 ERROR HANDLING 2-5
2.6.1 CALL-BY-REFERENCE 2-5
2.6.2 CALL-BY-VALUE 2-6
2.7 RELIABILITY AND PERFORMANCE 2-6
2.8 MATHEMATICAL FUNCTION SPECIFICATIONS 2-6
2.8.1 ABS 2-8
2.8.2 ACOS 2-9
2.8.3 AIMAG 2-10
2.8.4 AINT 2-11
2.8.5 ALOG 2-12
2.8.6 ALOG10 2-13
2.8.7 AMOD 2-14
2.8.8 ANINT 2-15
2.8.9 ASIN 2-16
2.8.10 ATAN 2-17
2.8.11 ATAN2 2-18
2.8.12 ATANH 2-19
2.8.13 CABS 2-20
2.8.14 CCOS 2-21
2.8.15 CEXP 2-22
2.8.16 CLOG 2-23

2.8.17 CONJG 2-24
2.8.18 COS 2-25
2.8.19 COSD 2-26
2.8.20 COSH 2-27
2.8.21 COTAN 2-28
2.8.22 CSIN 2-29
2.8.23 CSQRT 2-30
2.8.24 DABS 2-31
2.8.25 DACOS 2-32
2.8.26 DASIN 2-33
2.8.27 DATAN 2-34
2.8.28 DATAN2 2-35
2.8.29 DCOS 2-36
2.8.30 DCOSH 2-37
2.8.31 DDIM 2-38
2.8.32 DEXP 2-39
2.8.33 DIM 2-40
2.8.34 DINT 2-41

2.8.35 DLOG 2-42
2.8.36 DLOG10 2-43
2.8.37 DMOD 2-44
2.8.38 DNINT 2-45
2.8.39 DPROD 2-46
2.8.40 DSIGN 2-47
2.8.41 DSIN 2-48
2.8.42 DSINH 2-49
2.8.43 DSQRT 2-50
2.8.44 DTAN 2-51
2.8.45 DTANH 2-52
2.8.46 DTOD 2-53
2.8.47 DTOI 2-54
2.8.48 DTOX 2-55
2.8.49 DTOZ 2-56
2.8.50 ERF 2-57
2.8.51 ERFC 2-58
2.8.52 EXP 2-59
2.8.53 EXTB 2-60
2.8.54 IABS 2-61
2.8.55 IDIM 2-62
2.8.56 IDNINT 2-63
2.8.57 INSB 2-64
2.8.58 ISIGN 2-65
2.8.59 ITOD 2-66
2.8.60 ITOI 2-67
2.8.61 ITOX 2-68
2.8.62 ITOZ 2-69
2.8.63 MOD 2-70
2.8.64 NINT 2-71
2.8.65 RANF 2-72
2.8.66 RANGET 2-73
2.8.67 RANSET 2-74
2.8.68 SIGN 2-75
2.8.69 SIN 2-76
2.8.70 SIND 2-77

2.8.71 SINH 2-78
2.8.72 SQRT 2-79
2.8.73 TAN 2-80
2.8.74 TAND 2-81
2.8.75 TANH 2-82
2.8.76 XTOD 2-83
2.8.77 XTOI 2-84
2.8.78 XTOX 2-85
2.8.79 XTOZ 2-86
2.8.80 ZTOD 2-87
2.8.81 ZTOI 2-88
2.8.82 ZTOX 2-89
2.8.83 ZTOZ 2-90

3.0 NUMERIC CONVERSION AND ASSEMBLY LANGUAGE SUPPORT ROUTINES . 3-1
3.1 INTRODUCTION 3-1
3.2 DOCUMENTATION CONVENTIONS 3-1

3.3 LINKAGE INTERFACE 3-1
3.4 ERROR HANDLING 3-2
3.5 CONVERSION AND ALSS ROUTINE SPECIFICATIONS 3-2
3.5.1 MLP$BDP_CONVERSION 3-3
3.5.2 MLP$BDP_TO_BITS AND MLP$BITS_TO_BDP 3-4
3.5.3 MLP$COMPARE_BDP 3-5
3.5.4 MLP$COMPARE_BYTES 3-6
3.5.5 MLP$COMPARE_COLLATED 3-7
3.5.6 MLP$COMPARE_FLOATING 3-8
3.5.7 MLP$COMPUTE_FLOATING_NUMBER 3-9
3.5.8 MLP$CONVERT_FLOAT_TO_INTEGER 3-10
3.5.9 MLP$CONVERT_INTEGER_TO_FLOAT 3-11
3.5.10 MLP$INPUT_BASE_NUMBER 3-12
3.5.11 MLP$INPUT_FLOATING_MANTISSA 3-13
3.5.12 MLP$INPUT_FLOATING_NUMBER 3-14
3.5.13 MLP$INPUT_INTEGER 3-15
3.5.14 MLP$INPUT_UNPACKED_DECIMAL 3-16
3.5.15 MLP$MOVE_BYTES 3-17
3.5.16 MLP$OUTPUT_BASE_NUMBER 3-18
3.5.17 MLP$OUTPUT_FLOATING_DIGITS 3-19
3.5.18 MLP$OUTPUT_FLOATING_NUMBER 3-20
3.5.19 MLP$OUTPUT_INTEGER 3-22
3.5.20 MLP$ROUND_FLOATING_NUMBER 3-23
3.5.21 MLP$SCALE_FLOATING_NUMBER 3-24
3.5.22 MLP$SCAN_BYTES 3-25
3.5.23 MLP$TEST_FOR_EXCEPTION 3-26
3.5.24 MLP$TRANSLATE_BYTES 3-27
3.5.25 MLP$VAX_TO_180_FLOATING 3-28
3.5.26 MLP$VAX_TO_180_FORTRAN_LOGICAL 3-30
3.5.27 MLP$VAX_TO_180_INTEGER 3-31
3.5.28 MLP$VAX_TO_180_PACKED_DECIMAL 3-32
3.5.29 MLP$170_TO_180_BINARY 3-33
3.5.30 MLP$170_TO_180_FLOATING 3-34
3.5.31 MLP$170_TO_180_INTEGER 3-35
3.5.32 MLP$180_TO_170_BINARY 3-36
3.5.33 MLP$180_TO_170_FLOATING 3-37
3.5.34 MLP$180_TO_170_INTEGER 3-38

APPENDIX A - TYPES AND CONSTANTS FOR SUPPORT ROUTINES
. . . . A-1

A1.0 TYPES AND CONSTANTS FOR SUPPORT ROUTINES A1-1
A1.1 MLT$BDP_LENGTH A1-1
A1.2 MLT$BDP_TYPE A1-1
A1.3 MLT$COMPARE A1-1
A1.4 MLT$DIGIT_STRING_LENGTH A1-2
A1.5 MLT$ERROR A1-2
A1.6 MLT$EXPONENT_STYLE A1-2
A1.7 MLT$FLOATING_INPUT A1-2
A1.8 MLT$FLOATING_LENGTH A1-3
A1.9 MLT$FORMAT A1-3
A1.10 MLT$FORTRAN_LOGICAL_LENGTH A1-3
A1.11 MLT$HANDLE_BLANKS A1-3

A1.12 MLT$INTEGER_LENGTH A1-4
A1.13 MLT$INTEGER_TYPE A1-4
A1.14 MLT$JUSTIFY A1-4
A1.15 MLT$NON_DECIMAL_BASE A1-4
A1.16 MLT$OUTPUT_FORMAT A1-5
A1.17 MLT$SIGN_TREATMENT A1-5
A1.18 MLT$STRING_LENGTH A1-5
A1.19 MLT$VAX_FLOATING_TYPE A1-6
A1.20 MLT$VAX_INTEGER_LENGTH A1-6
A1.21 MLT$VAX_LOGICAL_LENGTH A1-6
A1.22 MLT$VAX_PACKED_DECIMAL_LENGTH A1-6

APPENDIX B - CMML MATHEMATICAL ERRORS
. . . . B-1

B1.0 CMML MATHEMATICAL ERRORS B1-1

APPENDIX C - MADIFY TO SCU CONVERSION
. . . . C-1

C1.0 MADIFY TO SCU CONVERSION C1-1