AD NUMBER

ADB120258

NEW LIMITATION CHANGE

TO

Approved for public release, distribution unlimited

FROM

Distribution authorized to U.S. Gov't. agencies and their contractors; Critical Technology; JUL 1987. Other requests shall be referred to Air Force Armament Laboratory, ATTN: FXG, Eglin AFB, FL 32542-5434.

AUTHORITY

HQ AFSC/MNOI WL ltr dtd 13 Feb 1992

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Distribution authorized to U.S. Government Agencies and their contractors; CT (over)
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFATL-TR-88-18, Vol 11
6a. NAME OF PERFORMING ORGANIZATION McDonnell Douglas Astronautics Company 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Aeromechanics Division
6c. ADDRESS (City, State, and ZIP Code) P.O. Box 516 St. Louis, MO 63166	7b. ADDRESS (City, State, and ZIP Code) Air Force Armament Laboratory Eglin AFB, FL 32542-5434
8a. NAME OF FUNDING/SPONSORING ORGANIZATION STARS Joint Program Office 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F08635-86-C-0025
8c. ADDRESS (City, State, and ZIP Code) Room 3D139 (1211 Fern St) The Pentagon Washington DC 20301-3081	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO.: 63756D; PROJECT NO.: 921C; TASK NO.: GZ; WORK UNIT ACCESSION NO.: 57

11. TITLE (Include Security Classification)
Common Ada Missile Package (CAMP) Project: Missile Software Parts, Vol 11: Detail Design Documents (Vol 7-12)

12. PERSONAL AUTHOR(S)
D. McNicholl, S. Cohen, C. Palmer, et al.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical Note	FROM Sep 85 TO Mar 88	March 1988	296

16. SUPPLEMENTARY NOTATION
SUBJECT TO EXPORT CONTROL LAWS.
Availability of this report is specified on verso of front cover. (over)

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Reusable Software, Missile Software, Software Generators Ada, Parts Composition, Systems, Software Parts

AD-B120 258

19. ABSTRACT (Continue on reverse if necessary and identify by block number)
The objective of the CAMP program is to demonstrate the feasibility of reusable Ada software parts in a real-time embedded application area; the domain chosen for the demonstration was that of missile flight software systems. This required that the existence of commonality within that domain be verified (in order to justify the development of parts for that domain), and that software parts be designed which address those areas identified. An associated parts system was developed to support parts usage. This Volume 1 of this document is the User's Guide to the CAMP Software parts; Volume 2 is the Version Description Document; Volume 3 is the Software Product Specification; Volumes 4-6 contain the Top-Level Design Document; and, Volumes 7-12 contain the Detail Design Documents.

part of

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Christine Anderson	22b. TELEPHONE (Include Area Code) (904) 882-2961
	22c. OFFICE SYMBOL AFATL/FXG

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

3. DISTRIBUTION/AVAILABILITY OF REPORT (CONCLUDED)

this report documents test and evaluation; distribution limitation applied March 1988. Other requests for this document must be referred to AFATL/FXG, Eglin AFB, Florida 32542-5434.

16. SUPPLEMENTARY NOTATION (CONCLUDED)

These technical notes accompany the CAMP final report AFATL-TR-85-93 (3 Vols)

AFATL-TR-88-18, Vol II

SOFTWARE DETAILED DESIGN DOCUMENT

FOR THE

MISSILE SOFTWARE PARTS

OF THE

COMMON ADA MISSILE PACKAGE (CAMP) PROJECT

CONTRACT F08635-86-C-0025

CDRL SEQUENCE NO. C007

30 OCTOBER 1987

AIR FORCE ARMAMENT LABORATORY
Air Force Systems Command ▮ United States Air Force ▮ Eglin Air Force Base, Florida

3.3.6.8 POLYNOMIALS (PACKAGE BODY) TLCSC P688 (CATALOG #P722-0)

This part is a package of packages. It contains specifications for all the polynomial functions required by the rest of the CAMP parts. Each subpackage, except General_Polynomial, contains function(s) for one type of polynomial (i.e. Hastings, Taylor series, etc.). These parts provide standard mathematical functions such as trigonometric and square root functions.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.1 REQUIREMENTS ALLOCATION

Name	Type	Requirements Allocation
Chebyshev	package	R214
Continued_Fractions	package	
Fike	package	R215
Hart	package	R216
Hastings	package	R217
Modified_ Newton_Raphson	package	R220
Newton_Raphson	package	R221
Taylor_Series	package	R222
General_Polynomial	package	partially meets CAMP requirements R214 thru R222
System_Functions	package	R223

3.3.6.8.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.3 INPUT/OUTPUT

None.

3.3.6.8.4 LOCAL DATA

None.

3.3.6.8.5 PROCESS CONTROL

Not applicable.

3.3.6.8.6 PROCESSING

The following describes the processing performed by this part:

```
with Math_Lib;
package body Polynomials is

   package body Chebyshev is separate;

   package body Cody_Waite is separate;

   package body Continued_Fractions is separate;

   package body Fike is separate;

   package body General_Polynomial is separate;

   package body Hart is separate;

   package body Hastings is separate;

   package body Modified_Newton_Raphson is separate;

   package body Newton_Raphson is separate;

   package body System_Functions is separate;

   package body Taylor_Series is separate;

   package body Reduction_Operations is separate;

end Polynomials;
```

3.3.6.8.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.8 LIMITATIONS

None.

3.3.6.8.9 LLCSC DESIGN

3.3.6.8.9.1 CHEBYSHEV PACKAGE DESIGN (CATALOG #P723-0)

This package contains a generic packages providing polynomial solutions to the sine function with inputs of Radians, Degrees, or Semicircles.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R214.

3.3.6.8.9.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.1.3 INPUT/OUTPUT

None.

3.3.6.8.9.1.4 LOCAL DATA

None.

3.3.6.8.9.1.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.1.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials)
package body Chebyshev is

   package body Chebyshev_Radian_Operations is separate;

   package body Chebyshev_Degree_Operations is separate;

   package body Chebyshev_Semicircle_Operations is separate;

end Chebyshev;
```

3.3.6.8.9.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.1.8 LIMITATIONS

None.

3.3.6.8.9.1.9 LLCSC DESIGN

3.3.6.8.9.1.9.1 CHEBYSHEV_RADIAN_OPERATIONS PACKAGE DESIGN (CATALOG #P724-0)

This generic package contains functions providing a Chebyshev polynomial solution for the sine function. Provisions are made for the function to handle units of radians. Outputs are of type sin_cos_ratio.

The following table lists the catalog numbers for subunits contained in this part:

Name	Catalog _#
Sin_R_5term	P725-0

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.1.9.1.1 REQUIREMENTS ALLOCATION

This part partially meets CAMP requirement R214.

3.3.6.8.9.1.9.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.1.9.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Radians	Floating point	Allows floating point representation of radian measurements.
Real	Floating point	General floating point representation.
Sin_Cos_Ratio	Floating point	Represents sines and cosines.
Tan_Ratio	Floating point	Represents tangent values.

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
One_Over_Pi	Radians	constant	constant value of inverse of Pi

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Overloaded operator to multiply radians * radians yielding a real result.

FORMAL PARAMETERS:

The following table describes the formal parameters for the functions contained in this part:

Function	Name	Type	Description
Sin_R_5term	Input	Radians	Input of angle for sine calculation

3.3.6.8.9.1.9.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Sin_R_C0	constant	1.34752_631	Coefficient for calculation
Sin_R_C1	constant	-1.55659_125	Coefficient for calculation
Sin_R_C2	constant	0.22275_7911	Coefficient for calculation
Sin_R_C3	constant	-0.01419_31743	Coefficient for calculation
Sin_R_C4	constant	0.00051_19072_74	Coefficient for calculation
Sin_R_B5	constant	-0.00001_18935_046	Coefficient for calculation

3.3.6.8.9.1.9.1.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.1.9.1.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials.Chebyshev)
package body Chebyshev_Radian_Operations is

   Sin_R_C0 : constant :=  1.34752_631;
   Sin_R_C1 : constant := -1.55659_125;
   Sin_R_C2 : constant :=  0.22275_7911;
   Sin_R_C3 : constant := -0.01419_31743;
   Sin_R_C4 : constant :=  0.00051_19072_74;
   Sin_R_B5 : constant := -0.00001_18935_046;

   function Sin_R_5term(Input : Radians) return Sin_Cos_Ratio is

      Inter_Result_4 : Real;
      Inter_Result_3 : Real;
      Inter_Result_2 : Real;
      Inter_Result_1 : Real;
      Inter_Result_0 : Real;
      Result         : Sin_Cos_Ratio;
      Y              : Real;
      Y_squared      : Real;

   begin
      Y := Input * One_Over_Pi;       -- converts radians to semicircles
      Y_Squared := Y * Y;
      Inter_Result_0 := 4.0 * Y_Squared - 2.0;
      Inter_Result_4 := Inter_Result_0 * Sin_R_B5 + Sin_R_C4;
      Inter_Result_3 := Inter_Result_0 * Inter_Result_4 - Sin_R_B5 +
                        Sin_R_C3;
      Inter_Result_2 := Inter_Result_0 * Inter_Result_3 - Inter_Result_4 +
                        Sin_R_C2;
      Inter_Result_1 := Inter_Result_0 * Inter_Result_2 - Inter_Result_3 +
                        Sin_R_C1;
      Inter_Result_0 := Inter_Result_0 * Inter_Result_1 - Inter_Result_2 +
                        Sin_R_C0;
      Inter_Result_0 := (Inter_Result_0 - (2.0 * Y_Squared - 1.0) *
                         Inter_Result_1) * Y;
      if Inter_Result_0 > 1.0 then
         Inter_Result_0 := 1.0;
      elsif Inter_Result_0 < -1.0 then
         Inter_Result_0 := -1.0;
      end if;
      Result := Sin_Cos_Ratio(Inter_Result_0);
      return Result;
   end Sin_R_5term;

end Chebyshev_Radian_Operations;
```

3.3.6.8.9.1.9.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.1.9.1.8 LIMITATIONS

None.

3.3.6.8.9.1.9.1.9 LLCSC DESIGN

None.

3.3.6.8.9.1.9.1.10 UNIT DESIGN

None.

3.3.6.8.9.1.9.2 CHEBYSHEV_DEGREE_OPERATIONS PACKAGE DESIGN (CATALOG #P1088-0)

This generic package contains functions providing a Chebyshev polynomial solution for the sine function. Provisions are made for the function to handle units of degrees. Outputs are of type sin_cos_ratio.

The following table lists the catalog numbers for subunits contained in this part:

Name	Catalog _#
Sin_D_5term	P727-0

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.1.9.2.1 REQUIREMENTS ALLOCATION

This part partially meets CAMP requirement R214.

3.3.6.8.9.1.9.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.1.9.2.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Degrees	Floating point	Allows floating point representation of degree measurements.
Real	Floating point	General floating point representation.
Sin_Cos_Ratio	Floating point	Represents sines and cosines.
Tan_Ratio	Floating point	Represents tangent values.

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
One_Over_Pi	Degrees	constant	constant value of inverse of Pi

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Overloaded operator to multiply degrees * degrees yielding a real result.

FORMAL PARAMETERS:

The following table describes the formal parameters for the functions contained in this part:

Function	Name	Type	Description
Sin_D_5term	Input	Degrees	Input of angle for sine calculation

3.3.6.8.9.1.9.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Sin_D_C0	constant	1.34752_631	Coefficient for calculation
Sin_D_C1	constant	-1.55659_125	Coefficient for calculation
Sin_D_C2	constant	0.22275_7911	Coefficient for calculation
Sin_D_C3	constant	-0.01419_31743	Coefficient for calculation
Sin_D_C4	constant	0.00051_19072_74	Coefficient for calculation
Sin_D_B5	constant	-0.00001_18935_046	Coefficient for calculation
One_Over_180	constant	0.00555_55555_5	Conversion constant

3.3.6.8.9.1.9.2.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.1.9.2.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials.Chebyshev)
package body Chebyshev_Degree_Operations is

   Sin_D_C0 : constant :=  1.34752_631;
   Sin_D_C1 : constant := -1.55659_125;
   Sin_D_C2 : constant :=  0.22275_7911;
   Sin_D_C3 : constant := -0.01419_31743;
   Sin_D_C4 : constant :=  0.00051_19072_74;
   Sin_D_B5 : constant := -0.00001_18935_046;

   One_Over_180   : constant :=  0.00555555555;

   function Sin_D_5term(Input : Degrees) return Sin_Cos_Ratio is

      Inter_Result_4 : Real;
      Inter_Result_3 : Real;
      Inter_Result_2 : Real;
      Inter_Result_1 : Real;
      Inter_Result_0 : Real;
      Result         : Sin_Cos_Ratio;
      Y              : Real;
      Y_squared      : Real;

   begin
      Y := Input * One_Over_180;       -- converts degrees to semicircles
      Y_Squared := Y * Y;
      Inter_Result_0 := 4.0 * Y_Squared - 2.0;
      Inter_Result_4 := Inter_Result_0 * Sin_D_B5 + Sin_D_C4;
      Inter_Result_3 := Inter_Result_0 * Inter_Result_4 - Sin_D_B5
                        + Sin_D_C3;
      Inter_Result_2 := Inter_Result_0 * Inter_Result_3 - Inter_Result_4
                        + Sin_D_C2;
      Inter_Result_1 := Inter_Result_0 * Inter_Result_2 - Inter_Result_3
                        + Sin_D_C1;
      Inter_Result_0 := Inter_Result_0 * Inter_Result_1 - Inter_Result_2
```

```
                    + Sin_D_C0;
      Inter_Result_0 := (Inter_Result_0 - (2.0 * Y_Squared - 1.0) *
                         Inter_Result_1) * Y;
      if Inter_Result_0 > 1.0 then
         Inter_Result_0 := 1.0;
      elsif Inter_Result_0 < -1.0 then
         Inter_Result_0 := -1.0;
      end if;
      Result := Sin_Cos_Ratio(Inter_Result_0);
      return Result;
   end Sin_D_5term;

end Chebyshev_Degree_Operations;
```

3.3.6.8.9.1.9.2.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.1.9.2.8 LIMITATIONS

None.

3.3.6.8.9.1.9.2.9 LLCSC DESIGN

None.

3.3.6.8.9.1.9.2.10 UNIT DESIGN

None.

3.3.6.8.9.1.9.3 CHEBYSHEV_SEMICIRCLE_OPERATIONS PACKAGE DESIGN (CATALOG #P728-0)

This generic package contains functions providing a Chebyshev polynomial solution for the sine function. Provisions are made for the function to handle units of semicircles. Outputs are of type sin_cos_ratio.

The following table lists the catalog numbers for subunits contained in this part:

Name	Catalog _#
Sin_S_5term	P729-0

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.1.9.3.1 REQUIREMENTS ALLOCATION

This part partially meets CAMP requirement R214.

3.3.6.8.9.1.9.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.1.9.3.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Semicircles	Floating point	Allows floating point representation of semicircle measurements.
Real	Floating point	General floating point representation.
Sin_Cos_Ratio	Floating point	Represents sines and cosines.
Tan_Ratio	Floating point	Represents tangent values.

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
One_Over_Pi	Semicircles	constant	constant value of inverse of Pi

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Overloaded operator to multiply semicircles * semicircles yielding a real result.

FORMAL PARAMETERS:

The following table describes the formal parameters for the functions contained in this part:

Function	Name	Type	Description
Sin_S_5term	Input	Semicircles	Input of angle for sine calculation

3.3.6.8.9.1.9.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Sin_S_C0	constant	1.34752_631	Coefficient for calculation
Sin_S_C1	constant	-1.55659_125	Coefficient for calculation
Sin_S_C2	constant	0.22275_7911	Coefficient for calculation
Sin_S_C3	constant	-0.01419_31743	Coefficient for calculation
Sin_S_C4	constant	0.00051_19072_74	Coefficient for calculation
Sin_S_B5	constant	-0.00001_18935_046	Coefficient for calculation

3.3.6.8.9.1.9.3.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.1.9.3.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials.Chebyshev)
package body Chebyshev_Semicircle_Operations is

   Sin_S_C0 : constant :=  1.34752_631;
   Sin_S_C1 : constant := -1.55659_125;
   Sin_S_C2 : constant :=  0.22275_7911;
   Sin_S_C3 : constant := -0.01419_31743;
   Sin_S_C4 : constant :=  0.00051_19072_74;
   Sin_S_B5 : constant := -0.00001_18935_046;

   function Sin_S_5term(Input : Semicircles) return Sin_Cos_Ratio is

      Inter_Result_4 : Real;
      Inter_Result_3 : Real;
      Inter_Result_2 : Real;
      Inter_Result_1 : Real;
      Inter_Result_0 : Real;
      Result         : Sin_Cos_Ratio;
      Y_squared      : Real;

   begin
      Y_Squared := Input * Input;
      Inter_Result_0 := 4.0 * Y_Squared - 2.0;
```

```
      Inter_Result_4 := Inter_Result_0 * Sin_S_B5 + Sin_S_C4;
      Inter_Result_3 := Inter_Result_0 * Inter_Result_4 - Sin_S_B5 +
                        Sin_S_C3;
      Inter_Result_2 := Inter_Result_0 * Inter_Result_3 - Inter_Result_4 +
                        Sin_S_C2;
      Inter_Result_1 := Inter_Result_0 * Inter_Result_2 - Inter_Result_3 +
                        Sin_S_C1;
      Inter_Result_0 := Inter_Result_0 * Inter_Result_1 - Inter_Result_2 +
                        Sin_S_C0;
      Inter_Result_0 := (Inter_Result_0 - (2.0 * Y_Squared - 1.0) *
                        Inter_Result_1) * Real(Input);
      if Inter_Result_0 > 1.0 then
         Inter_Result_0 := 1.0;
      elsif Inter_Result_0 < -1.0 then
         Inter_Result_0 := -1.0;
      end if;
      Result := Sin_Cos_Ratio(Inter_Result_0);
      return Result;
   end Sin_S_5term;

end Chebyshev_Semicircle_Operations;
```

3.3.6.8.9.1.9.3.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.1.9.3.8 LIMITATIONS

None.

3.3.6.8.9.1.9.3.9 LLCSC DESIGN

None.

3.3.6.8.9.1.9.3.10 UNIT DESIGN

None.

3.3.6.8.9.1.10 UNIT DESIGN

None.

(This page left intentionally blank.)

3.3.6.8.9.2 FIKE PACKAGE DESIGN (CATALOG #P734-0)

This package contains a generic package providing a Fike polynomial solution for the arcsine function.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R215.

3.3.6.8.9.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.2.3 INPUT/OUTPUT

None.

3.3.6.8.9.2.4 LOCAL DATA

None.

3.3.6.8.9.2.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.2.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials)
package body Fike is

   package body Fike_Semicircle_Operations is separate;

end Fike;
```

3.3.6.8.9.2.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.2.8 LIMITATIONS

None.

3.3.6.8.9.2.9 LLCSC DESIGN

3.3.6.8.9.2.9.1 FIKE_SEMICIRCLE_OPERATIONS PACKAGE DESIGN (CATALOG #P735-0)

This generic package contains a function providing a Fike polynomial solution for the arcsine function. This package is designed to handle units of semicircles.

The following table lists the catalog numbers for subunits contained in this part:

Name	Catalog _#
Arcsin_S_6term	P736-0
Arccos_S_6term	P737-0

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.2.9.1.1 REQUIREMENTS ALLOCATION

This part partially meets CAMP requirement R215.

3.3.6.8.9.2.9.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.2.9.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Semicircles	Floating point	Allows floating point representation of semicircle measurements.
Real	Floating point	General floating point representation.
Sin_Cos_Ratio	Floating point	Represents sines and cosines.

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
Sqrt	funtion	returns the square root of type real

FORMAL PARAMETERS:

The following table describes the formal parameters for the functions contained in this part:

Function	Name	Type	Description
Arcsin_S_6term	Input	Sin_Cos_Ratio	Input for arcsine computation

3.3.6.8.9.2.9.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Arcsin_C1	constant	0.31830_99886	Coefficient for term 1
Arcsin_C3	constant	0.05305_20148	Coefficient for term 3
Arcsin_C5	constant	0.02385_63606	Coefficient for term 5
Arcsin_C7	constant	0.01448_96675	Coefficient for term 7
Arcsin_C9	constant	0.00763_75322_8	Coefficient for term 9
Arcsin_C11	constant	0.01350_18593	Coefficient for term 11

3.3.6.8.9.2.9.1.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.2.9.1.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials.Fike)
package body Fike_Semicircle_Operations is

   Arcsin_C1  : constant := 0.31830_9886;
   Arcsin_C3  : constant := 0.05305_20148;
   Arcsin_C5  : constant := 0.02385_63606;
   Arcsin_C7  : constant := 0.01448_96675;
   Arcsin_C9  : constant := 0.00763_75322_8;
   Arcsin_C11 : constant := 0.01350_18593;

   function Arcsin_S_6term (Input : Sin_Cos_Ratio) return Semicircles is
```

```
      Input_Squared : Real;
      Inter_Result  : Real;
      Left_Quadrant : Boolean;
      Mod_Input     : Real;
      Result        : Semicircles;

   begin
      if Abs( Input ) > 0.5 then
         Mod_Input := Sqrt( Real((1.0 - Abs(Input)) * 0.5) );
         Left_Quadrant := True;
      else
         Mod_Input := Real(Input);
         Left_Quadrant := False;
      end if;
      Input_Squared := Mod_Input * Mod_Input;
      Inter_Result := (((((Arcsin_C11 * Input_Squared +
                     Arcsin_C9) * Input_Squared +
                     Arcsin_C7) * Input_Squared +
                     Arcsin_C5) * Input_Squared +
                     Arcsin_C3) * Input_Squared +
                     Arcsin_C1) * Mod_Input;
      if Left_Quadrant then
         if Input > 0.0 then
            Inter_Result := 0.5 - (2.0 * Inter_Result);
         else
            Inter_Result := - (0.5 - (2.0 * Inter_Result));
         end if;
      end if;
      Result := Semicircles(Inter_Result);
      return Result;
   end Arcsin_S_6term;

   function Arccos_S_6term (Input : Sin_Cos_Ratio) return Semicircles is

      Input_Squared : Real;
      Inter_Result  : Real;
      Left_Quadrant : Boolean;
      Mod_Input     : Real;
      Result        : Semicircles;

   begin
      if Abs( Input ) > 0.5 then
         Mod_Input := Sqrt( Real((1.0 - Abs(Input)) * 0.5) );
         Left_Quadrant := True;
      else
         Mod_Input := Real(Input);
         Left_Quadrant := False;
      end if;
      Input_Squared := Mod_Input * Mod_Input;
      Inter_Result := (((((Arcsin_C11 * Input_Squared +
                     Arcsin_C9) * Input_Squared +
                     Arcsin_C7) * Input_Squared +
                     Arcsin_C5) * Input_Squared +
                     Arcsin_C3) * Input_Squared +
                     Arcsin_C1) * Mod_Input;
      if Left_Quadrant then
         if Input > 0.0 then
```

```
            Inter_Result := 0.5 - (2.0 * Inter_Result);
         else
            Inter_Result := - (0.5 - (2.0 * Inter_Result));
         end if;
      end if;
      Result := Semicircles(Inter_Result);
--    -- convert to Arccos by applying formula
--    -- Arccosine = Pi/2 - Arcsine
      Result := 0.5 - Result;
      return Result;
   end Arccos_S_6term;

end Fike_Semicircle_Operations;
```

3.3.6.8.9.2.9.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.2.9.1.8 LIMITATIONS

None.

3.3.6.8.9.2.9.1.9 LLCSC DESIGN

None.

3.3.6.8.9.2.9.1.10 UNIT DESIGN

None.

3.3.6.8.9.2.10 UNIT DESIGN

None.

(This page left intentionally blank.)

3.3.6.8.9.3 HART PACKAGE DESIGN (CATALOG #P740-0)

This packages contains generic packages providing Hart a polynomial solution for the cosine function. Provisions are made for the cosine function to handle units of radians or degrees, respectively. Outputs may be of type sin_cos_-ratio.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R216.

3.3.6.8.9.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.3.3 INPUT/OUTPUT

None.

3.3.6.8.9.3.4 LOCAL DATA

None.

3.3.6.8.9.3.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.3.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials)
package body Hart is

   package body Hart_Radian_Operations is separate;

   package body Hart_Degree_Operations is separate;

end Hart;
```

3.3.6.8.9.3.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.3.8 LIMITATIONS

None.

3.3.6.8.9.3.9 LLCSC DESIGN

3.3.6.8.9.3.9.1 HART_RADIAN_OPERATIONS PACKAGE DESIGN (CATALOG #P741-0)

This generic package contains a function providing a Hart polynomial solution for the cosine function. This package is designed to accept inputs in terms of radians.

The following table lists the catalog numbers for subunits contained in this part:

Name	Catalog _#
Cos_R_5term	P742-0

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.3.9.1.1 REQUIREMENTS ALLOCATION

This part partially meets CAMP requirement R216.

3.3.6.8.9.3.9.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.3.9.1.3 INPUT/OUTPUT

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Radians	Floating point	Allows floating point representation of radian measurements.
Real	Floating point	General floating point representation.
Sin_Cos_Ratio	Floating point	Represents sines and cosines.

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Overloaded operator to multiply radians * radians yielding a real result.

GENERIC PARAMETERS:

FORMAL PARAMETERS:

The following table describes the formal parameters for the functions contained in this part:

Function	Name	Type	Description
Cos_R_5term	Input	Radians	Input angle for cosine function

3.3.6.8.9.3.9.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Cos_R_C0	constant	0.99999_9953	Coefficient for term 2
Cos_R_C2	constant	-0.49999_9905	Coefficient for term 4
Cos_R_C4	constant	0.04166_35846	Coefficient for term 6
Cos_R_C6	constant	-0.00138_53704_2	Coefficient for term 8
Cos_R_C8	constant	0.00002_31539_316	Coefficient for term 10

3.3.6.8.9.3.9.1.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.3.9.1.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials.Hart)
package body Hart_Radian_Operations is

   Cos_R_C0  : constant :=  0.99999_9953;
   Cos_R_C2  : constant := -0.49999_9053;
   Cos_R_C4  : constant :=  0.04166_35847;
   Cos_R_C6  : constant := -0.00138_53704_3;
   Cos_R_C8  : constant :=  0.00002_31539_317;

   function Cos_R_5term (Input : Radians) return Sin_Cos_Ratio is
```

```
      Inter_Result : Real;
      Mod_Input    : Radians;
      Result       : Sin_Cos_Ratio;
      X_Squared    : Radians;

   begin

      if Input >= Pi_Over_2 then
         Mod_Input := Pi - Input;
      else
         Mod_Input := Input;
      end if;
      X_Squared := Mod_Input * Mod_Input;
      Inter_Result := (((Cos_R_C8  * X_Squared +
                         Cos_R_C6) * X_Squared +
                         Cos_R_C4) * X_Squared +
                         Cos_R_C2) * X_Squared;
      Inter_Result := Inter_Result + Cos_R_C0;
      if Input >= Pi_Over_2 then
         Inter_Result := - Inter_Result;
      end if;
      If Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;

   end Cos_R_5term;

end Hart_Radian_Operations;
```

3.3.6.8.9.3.9.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.3.9.1.8 LIMITATIONS

None.

3.3.6.8.9.3.9.1.9 LLCSC DESIGN

None.

3.3.6.8.9.3.9.1.10 UNIT DESIGN

None.

3.3.6.8.9.3.9.2 HART_DEGREE_OPERATIONS PACKAGE DESIGN (CATALOG #P743-0)

This generic package contains a function providing a Hart polynomial solution for the cosine function. This package is designed to accept inputs in terms of degrees.

The following table lists the catalog numbers for subunits contained in this part:

Name	Catalog _#
Cos_D_5term	P744-0

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.3.9.2.1 REQUIREMENTS ALLOCATION

This part partially meets CAMP requirement R216.

3.3.6.8.9.3.9.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.3.9.2.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Degrees	Floating point	Allows floating point representation of degree measurements.
Real	Floating point	General floating point representation.
Sin_Cos_Ratio	Floating point	Represents sines and cosines.

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Overloaded operator to multiply degrees * degrees yielding a real result.

FORMAL PARAMETERS:

The following table describes the formal parameters for the functions contained in this part:

Function	Name	Type	Description
Cos_D_5term	Input	Degrees	Input angle for cosine function

3.3.6.8.9.3.9.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Cos_D_C0	constant	0.99999_9953	Coefficient for term 2
Cos_D_C2	constant	-0.49999_9905	Coefficient for term 4
Cos_D_C4	constant	0.04166_35846	Coefficient for term 6
Cos_D_C6	constant	-0.00138_53704_2	Coefficient for term 8
Cos_D_C8	constant	0.00002_31539_316	Coefficient for term 10

3.3.6.8.9.3.9.2.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.3.9.2.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials.Hart)
package body Hart_Degree_Operations is

   Cos_D_C0 : constant :=   0.99999_9953;
   Cos_D_C2 : constant := - 1.52308_42e-04;
   Cos_D_C4 : constant :=   3.86603_79e-09;
   Cos_D_C6 : constant := - 3.91588_67e-14;
   Cos_D_C8 : constant :=   1.99362_60e-19;

   function Cos_D_5term (Input : Degrees) return Sin_Cos_Ratio is

      Inter_Result : Real;
      Mod_Input    : Degrees;
      Result       : Sin_Cos_Ratio;
      X_Squared    : Real;

   begin

      if Input >= 90.0 then
```

```
            Mod_Input := 180.0 - Input;
         else
            Mod_Input := Input;
         end if;
         X_Squared := Mod_Input * Mod_Input;
         Inter_Result := (((Cos_D_C8  * X_Squared +
                            Cos_D_C6) * X_Squared +
                            Cos_D_C4) * X_Squared +
                            Cos_D_C2) * X_Squared;
         Inter_Result := Inter_Result + Cos_D_C0;
         if Input >= 90.0 then
            Inter_Result := - Inter_Result;
         end if;
         If Inter_Result > 1.0 then
            Inter_Result := 1.0;
         elsif Inter_Result < -1.0 then
            Inter_Result := -1.0;
         end if;
         Result := Sin_Cos_Ratio( Inter_Result );
         return Result;

      end Cos_D_5term;

   end Hart_Degree_Operations;
```

3.3.6.8.9.3.9.2.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.3.9.2.8 LIMITATIONS

None.

3.3.6.8.9.3.9.2.9 LLCSC DESIGN

None.

3.3.6.8.9.3.9.2.10 UNIT DESIGN

None.

3.3.6.8.9.3.10 UNIT DESIGN

None.

(This page left intentionally blank.)

3.3.6.8.9.4 HASTINGS PACKAGE DESIGN (CATALOG #P745-0)

This packages contains generic functions providing Hastings polynomial solutions for a set of trigonometric functions, which include sine, cosine, tangent, and arctangent. Provisions are made for the trigonometric functions to handle units of radians or degrees.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R217.

3.3.6.8.9.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.4.3 INPUT/OUTPUT

None.

3.3.6.8.9.4.4 LOCAL DATA

None.

3.3.6.8.9.4.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.4.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials)
package body Hastings is

   package body Hastings_Radian_Operations is separate;

   package body Hastings_Degree_Operations is separate;

end Hastings;
```

3.3.6.8.9.4.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.4.8 LIMITATIONS

None.

3.3.6.8.9.4.9 LLCSC DESIGN

3.3.6.8.9.4.9.1 HASTINGS_RADIAN_OPERATIONS PACKAGE DESIGN (CATALOG #P746-0)

This packages contains generic functions providing Hastings polynomial solutions for a set of trigonometric functions. This package is designed to handle units of radians.

The following table lists the catalog numbers for subunits contained in this part:

Name	Catalog _#
Sin_R_5term	P747-0
Sin_R_4term	P748-0
Cos_R_5term	P749-0
Cos_R_4term	P750-0
Tan_R_5term	P751-0
Tan_R_4term	P752-0
Arctan_R_8term	P753-0
Arctan_R_7term	P754-0
Arctan_R_6term	P755-0
Mod_Arctan_R_8term	P756-0
Mod_Arctan_R_7term	P757-0
Mod_Arctan_R_6term	P758-0

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.4.9.1.1 REQUIREMENTS ALLOCATION

This part partially meets CAMP requirement R217.

3.3.6.8.9.4.9.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.4.9.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Radians	Floating point	Allows floating point representation of radian measurements.
Real	Floating point	General floating point representation.
Sin_Cos_Ratio	Floating point	Represents sines and cosines.
Tan_Ratio	Floating point	Represents tangent values.

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
Pi_Over_2	Radians	constant	constant value of Pi divided by 2
Pi_Over_4	Radians	constant	constant value of Pi divided by 4

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Overloaded operator to multiply radians * radians yielding a real result.

FORMAL PARAMETERS:

The following table describes the formal parameters for the functions contained in this part:

Function	Name	Type	Description
Sine Functions	Input	Radians	Input angle for sine computation
Cosine Functions	Input	Radians	Input angle for cosine computation
Tangent Functions	Input	Radians	Input angle for tangent computation
Arctangent Functions	Input	Tan_Ratio	Input for arctangent computation

3.3.6.8.9.4.9.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Sin_R_C1_5term	constant	0.99999_9994	1st term sine coefficient
Sin_R_C3_5term	constant	-0.16666_6566	3rd term sine coefficient
Sin_R_C5_5term	constant	0.00833_30251_7	5th term sine coefficient
Sin_R_C7_5term	constant	-0.00019_80741_43	7th term sine coefficient
Sin_R_C9_5term	constant	0.00000_26018_8690	9th term sine coefficient
Sin_R_C1_4term	constant	0.99999_9	1st term sine coefficient
Sin_R_C3_4term	constant	-0.16665_5	3rd term sine coefficient
Sin_R_C5_4term	constant	0.000831_190	5th term sine coefficient
Sin_R_C7_4term	constant	-0.00018_4882	7th term sine coefficient
Arctan_R_C1_8term	constant	0.99999_9333	1st term arctangent coeffiecient
Arctan_R_C3_8term	constant	-0.33329_8560	3rd term arctangent coeffiecient
Arctan_R_C5_8term	constant	0.19946_5360	5th term arctangent coeffiecient
Arctan_R_C7_8term	constant	-0.13908_5335	7th term arctangent coeffiecient
Arctan_R_C9_8term	constant	0.09642_0044	9th term arctangent coeffiecient
Arctan_R_C11_8term	constant	-0.05590_9886	11th term arctangent coeffiecient
Arctan_R_C13_8term	constant	0.02186_1229	13th term arctangent coeffiecient
Arctan_R_C15_8term	constant	-0.00405_4080	15th term arctangent coeffiecient
Arctan_R_C1_7term	constant	0.99999_6115	1st term arctangent coeffiecient
Arctan_R_C3_7term	constant	-0.33317_3758	3rd term arctangent coeffiecient
Arctan_R_C5_7term	constant	0.19807_8690	5th term arctangent coeffiecient
Arctan_R_C7_7term	constant	-0.13233_5096	7th term arctangent coeffiecient
Arctan_R_C9_7term	constant	0.07962_6318	9th term arctangent coeffiecient
Arctan_R_C11_7term	constant	-0.03360_6209	11th term arctangent coeffiecient
Arctan_R_C13_7term	constant	0.00681_2411	13th term arctangent coeffiecient
Arctan_R_C1_6term	constant	0.99997_726	1st term arctangent coeffiecient
Arctan_R_C3_6term	constant	-0.33262_347	3rd term arctangent coeffiecient
Arctan_R_C5_6term	constant	0.19354_346	5th term arctangent coeffiecient

Arctan_R_C7_6term	constant	-0.11643_287	7th term arctangent coeffiecient
Arctan_R_C9_6term	constant	0.05265_332	9th term arctangent coeffiecient
Arctan_R_C11_6term	constant	-0.01172_120	11th term arctangent coeffiecient

3.3.6.8.9.4.9.1.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.4.9.1.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials.Hastings)
package body Hastings_Radian_Operations is

   Sin_R_C1_5term : constant :=  0.99999_9995;
   Sin_R_C3_5term : constant := -0.16666_6567;
   Sin_R_C5_5term : constant :=  0.00833_30251_7;
   Sin_R_C7_5term : constant := -0.00019_80741_43;
   Sin_R_C9_5term : constant :=  0.00000_26018_8690;

   Sin_R_C1_4term : constant :=  0.99999_9;
   Sin_R_C3_4term : constant := -0.16665_5;
   Sin_R_C5_4term : constant :=  0.00831_190;
   Sin_R_C7_4term : constant := -0.00018_4882;

   Arctan_R_C1_8term  : constant :=  0.99999_9333;
   Arctan_R_C3_8term  : constant := -0.33329_8560;
   Arctan_R_C5_8term  : constant :=  0.19946_5360;
   Arctan_R_C7_8term  : constant := -0.13908_5335;
   Arctan_R_C9_8term  : constant :=  0.09642_00441;
   Arctan_R_C11_8term : constant := -0.05590_98861;
   Arctan_R_C13_8term : constant :=  0.02186_12288;
   Arctan_R_C15_8term : constant := -0.00405_40580;

   Arctan_R_C1_7term  : constant :=  0.99999_6115;
   Arctan_R_C3_7term  : constant := -0.33317_3758;
   Arctan_R_C5_7term  : constant :=  0.19807_8690;
   Arctan_R_C7_7term  : constant := -0.13233_5096;
   Arctan_R_C9_7term  : constant :=  0.07962_6318;
   Arctan_R_C11_7term : constant := -0.03360_6269;
   Arctan_R_C13_7term : constant :=  0.00681_2411;

   Arctan_R_C1_6term  : constant :=  0.99997_726;
   Arctan_R_C3_6term  : constant := -0.33262_347;
   Arctan_R_C5_6term  : constant :=  0.19354_346;
   Arctan_R_C7_6term  : constant := -0.11643_287;
   Arctan_R_C9_6term  : constant :=  0.05265_332;
   Arctan_R_C11_6term : constant := -0.01172_120;
```

```
-- --sine functions

   function Sin_R_5term(Input : Radians) return Sin_Cos_Ratio is

      Input_Squared : Real;
      Inter_Result  : Real;
      Result        : Sin_Cos_Ratio;

   begin

      Input_Squared := Input * Input;
      Inter_Result   :=             ((((Sin_R_C9_5term  *
                        Input_Squared + Sin_R_C7_5term) *
                        Input_Squared + Sin_R_C5_5term) *
                        Input_Squared + Sin_R_C3_5term) *
                        Input_Squared + Sin_R_C1_5term);
      Inter_Result := Inter_Result * Real(Input);
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;

   end Sin_R_5term;


   function Sin_R_4term(Input : Radians) return Sin_Cos_Ratio is

      Input_Squared : Real;
      Inter_Result  : Real;
      Result        : Sin_Cos_Ratio;

   begin

      Input_Squared := Input * Input;
      Inter_Result   :=              (((Sin_R_C7_4term  *
                        Input_Squared + Sin_R_C5_4term) *
                        Input_Squared + Sin_R_C3_4term) *
                        Input_Squared + Sin_R_C1_4term);
      Inter_Result := Inter_Result * Real(Input);
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;

   end Sin_R_4term;

-- --cosine functions


   function Cos_R_5term(Input : Radians) return Sin_Cos_Ratio is
```

```
      Input_Squared : Real;
      Inter_Result  : Real;
      Mod_Input     : Radians;
      Result        : Sin_Cos_Ratio;

   begin

      Mod_Input := Pi_Over_2 - Input;
      Input_Squared := Mod_Input * Mod_Input;
      Inter_Result   :=            ((((Sin_R_C9_5term  *
                        Input_Squared + Sin_R_C7_5term) *
                        Input_Squared + Sin_R_C5_5term) *
                        Input_Squared + Sin_R_C3_5term) *
                        Input_Squared + Sin_R_C1_5term);
      Inter_Result := Inter_Result * Real(Mod_Input);
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;

   end Cos_R_5term;


   function Cos_R_4term(Input : Radians) return Sin_Cos_Ratio is

      Input_Squared : Real;
      Inter_Result  : Real;
      Mod_Input     : Radians;
      Result        : Sin_Cos_Ratio;

   begin

      Mod_Input := Pi_Over_2 - Input;
      Input_Squared := Mod_Input * Mod_Input;
      Inter_Result   :=             (((Sin_R_C7_4term  *
                        Input_Squared + Sin_R_C5_4term) *
                        Input_Squared + Sin_R_C3_4term) *
                        Input_Squared + Sin_R_C1_4term);
      Inter_Result := Inter_Result * Real(Mod_Input);
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;

   end Cos_R_4term;

-- -- Tangent functions

   function Tan_R_5term (Input : Radians) return Tan_Ratio is
      Sin : Sin_Cos_Ratio;
```

```
      Cos : Sin_Cos_Ratio;
   begin
      Sin := Sin_R_5term(Input);
      if Input < 0.0 then
         Cos := - Cos_R_5term( Pi + Input );
      else
         Cos := Cos_R_5term(Input);
      end if;
      return Tan_Ratio(Sin / Cos);
   end Tan_R_5term;


   function Tan_R_4term (Input : Radians) return Tan_Ratio is
      Sin : Sin_Cos_Ratio;
      Cos : Sin_Cos_Ratio;
   begin
      Sin := Sin_R_4term(Input);
      if Input < 0.0 then
         Cos := - Cos_R_4term( Pi + Input );
      else
         Cos := Cos_R_4term(Input);
      end if;
      return Tan_Ratio(Sin / Cos);
   end Tan_R_4term;


-- -- Arctangent functions

   function Arctan_R_8term (Input : Tan_Ratio) return Radians is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Result        : Radians;

   begin
      Input_Squared := Input * Input;
      Inter_Result   :=          (((((((Arctan_R_C15_8term  *
                        Input_Squared + Arctan_R_C13_8term) *
                        Input_Squared + Arctan_R_C11_8term) *
                        Input_Squared + Arctan_R_C9_8term)  *
                        Input_Squared + Arctan_R_C7_8term)  *
                        Input_Squared + Arctan_R_C5_8term)  *
                        Input_Squared + Arctan_R_C3_8term)  *
                        Input_Squared + Arctan_R_C1_8term)  *
                        Input;
      Result := Radians(Inter_Result);
      return Result;
   end Arctan_R_8term;


   function Arctan_R_7term (Input : Tan_Ratio) return Radians is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Result        : Radians;

   begin
```

```
      Input_Squared := Input * Input;
      Inter_Result   :=           ((((((Arctan_R_C13_7term  *
                        Input_Squared + Arctan_R_C11_7term) *
                        Input_Squared + Arctan_R_C9_7term)  *
                        Input_Squared + Arctan_R_C7_7term)  *
                        Input_Squared + Arctan_R_C5_7term)  *
                        Input_Squared + Arctan_R_C3_7term)  *
                        Input_Squared + Arctan_R_C1_7term)  *
                        Input;
      Result := Radians(Inter_Result);
      return Result;
   end Arctan_R_7term;


   function Arctan_R_6term (Input : Tan_Ratio) return Radians is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Result        : Radians;

   begin
      Input_Squared := Input * Input;
      Inter_Result   :=           (((((Arctan_R_C11_6term  *
                        Input_Squared + Arctan_R_C9_6term)  *
                        Input_Squared + Arctan_R_C7_6term)  *
                        Input_Squared + Arctan_R_C5_6term)  *
                        Input_Squared + Arctan_R_C3_6term)  *
                        Input_Squared + Arctan_R_C1_6term)  *
                        Input;
      Result := Radians(Inter_Result);
      return Result;
   end Arctan_R_6term;


-- -- Modified Hastings Arctangent functions

   function Mod_Arctan_R_8term (Input : Tan_Ratio) return Radians is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Mod_Input     : Tan_Ratio;
      Result        : Radians;

   begin
      if Input >= 0.0 then
         Mod_Input := Input;
      else
         Mod_Input := - Input;
      end if;
      Mod_Input := (Mod_Input - 1.0) / (Mod_Input + 1.0);
      Input_Squared := Mod_Input * Mod_Input;
      Inter_Result   :=           ((((((Arctan_R_C15_8term  *
                        Input_Squared + Arctan_R_C13_8term) *
                        Input_Squared + Arctan_R_C11_8term) *
                        Input_Squared + Arctan_R_C9_8term)  *
                        Input_Squared + Arctan_R_C7_8term)  *
                        Input_Squared + Arctan_R_C5_8term)  *
```

```
                        Input_Squared + Arctan_R_C3_8term)  *
                        Input_Squared + Arctan_R_C1_8term)  *
                        Mod_Input;
      Result := Radians(Inter_Result) + Pi_Over_4;
      if Input < 0.0 then
         Result := - Result;
      end if;
      return Result;
   end Mod_Arctan_R_8term;


   function Mod_Arctan_R_7term (Input : Tan_Ratio) return Radians is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Mod_Input     : Tan_Ratio;
      Result        : Radians;

   begin
      if Input >= 0.0 then
         Mod_Input := Input;
      else
         Mod_Input := - Input;
      end if;
      Mod_Input := (Mod_Input - 1.0) / (Mod_Input + 1.0);
      Input_Squared := Mod_Input * Mod_Input;
      Inter_Result   :=           (((((Arctan_R_C13_7term  *
                        Input_Squared + Arctan_R_C11_7term) *
                        Input_Squared + Arctan_R_C9_7term)  *
                        Input_Squared + Arctan_R_C7_7term)  *
                        Input_Squared + Arctan_R_C5_7term)  *
                        Input_Squared + Arctan_R_C3_7term)  *
                        Input_Squared + Arctan_R_C1_7term)  *
                        Mod_Input;
      Result := Radians(Inter_Result) + Pi_Over_4;
      if Input < 0.0 then
         Result := - Result;
      end if;
      return Result;
   end Mod_Arctan_R_7term;


   function Mod_Arctan_R_6term (Input : Tan_Ratio) return Radians is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Mod_Input     : Tan_Ratio;
      Result        : Radians;

   begin
      if Input >= 0.0 then
         Mod_Input := Input;
      else
         Mod_Input := - Input;
      end if;
      Mod_Input := (Mod_Input - 1.0) / (Mod_Input + 1.0);
      Input_Squared := Mod_Input * Mod_Input;
```

```
      Inter_Result   :=            (((((Arctan_R_C11_6term  *
                        Input_Squared + Arctan_R_C9_6term)  *
                        Input_Squared + Arctan_R_C7_6term)  *
                        Input_Squared + Arctan_R_C5_6term)  *
                        Input_Squared + Arctan_R_C3_6term)  *
                        Input_Squared + Arctan_R_C1_6term)  *
                        Mod_Input;
      Result := Radians(Inter_Result) + Pi_Over_4;
      if Input < 0.0 then
         Result := - Result;
      end if;
      return Result;
   end Mod_Arctan_R_6term;


end Hastings_Radian_Operations;
```

3.3.6.8.9.4.9.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.4.9.1.8 LIMITATIONS

None.

3.3.6.8.9.4.9.1.9 LLCSC DESIGN

None.

3.3.6.8.9.4.9.1.10 UNIT DESIGN

None.

3.3.6.8.9.4.9.2 HASTINGS_DEGREE_OPERATIONS PACKAGE DESIGN (CATALOG #P759-0)

This generic package contains functions providing Hastings polynomial solutions for a set of trigonometric functions. This package is designed to handle units of degrees.

The following table lists the catalog numbers for subunits contained in this part:

Name	Catalog _#
Sin_D_5term	P760-0
Sin_D_4term	P761-0
Cos_D_5term	P762-0
Cos_D_4term	P763-0
Tan_D_5term	P764-0
Tan_D_4term	P765-0

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.4.9.2.1 REQUIREMENTS ALLOCATION

This part partially meets CAMP requirement R217.

3.3.6.8.9.4.9.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.4.9.2.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Degrees	Floating point	Allows floating point representation of degree measurements.
Real	Floating point	General floating point representation.
Sin_Cos_Ratio	Floating point	Represents sines and cosines.
Tan_Ratio	Floating point	Represents tangent values.

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
Pi	Degrees	constant	constant value of Pi

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Overloaded operator to multiply degrees * degrees yielding a real result.

FORMAL PARAMETERS:

The following table describes the formal parameters for the functions contained in this part:

Function	Name	Type	Description
Sine Functions	Input	Degrees	Input angle for sine computation
Cosine Functions	Input	Degrees	Input angle for cosine computation
Tangent Functions	Input	Degrees	Input angle for tangent computation

3.3.6.8.9.4.9.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Sin_D_C1_5term	constant	0.99999_9994	1st term sine coefficient
Sin_D_C3_5term	constant	-0.16666_6566	3rd term sine coefficient
Sin_D_C5_5term	constant	0.00833_30251_7	5th term sine coefficient
Sin_D_C7_5term	constant	-0.00019_80741_43	7th term sine coefficient
Sin_D_C9_5term	constant	0.00000_26018_8690	9th term sine coefficient
Sin_D_C1_4term	constant	0.99999_9	1st term sine coefficient
Sin_D_C3_4term	constant	-0.16665_5	3rd term sine coefficient
Sin_D_C5_4term	constant	0.000831_190	5th term sine coefficient
Sin_D_C7_4term	constant	-0.00018_4882	7th term sine coefficient

3.3.6.8.9.4.9.2.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.4.9.2.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials.Hastings)
package body Hastings_Degree_Operations is
```

```
      Sin_D_C1_5term : constant :=   1.74532_92e-02;
      Sin_D_C3_5term : constant := - 8.86095_625e-07;
      Sin_D_C5_5term : constant :=   1.34955_172e-11;
      Sin_D_C7_5term : constant := - 9.77168_260e-17;
      Sin_D_C9_5term : constant :=   3.91006_135e-22;

      Sin_D_C1_4term : constant :=   1.74533e-02;
      Sin_D_C3_4term : constant := - 8.86037e-07;
      Sin_D_C5_4term : constant :=   1.34613e-11;
      Sin_D_C7_4term : constant := - 9.12087e-17;

   -- --sine functions


      function Sin_D_5term(Input : Degrees) return Sin_Cos_Ratio is

         Input_Squared : Real;
         Inter_Result  : Real;
         Result        : Sin_Cos_Ratio;

      begin

         Input_Squared := Input * Input;
         Inter_Result   :=             ((((Sin_D_C9_5term  *
                         Input_Squared + Sin_D_C7_5term) *
                         Input_Squared + Sin_D_C5_5term) *
                         Input_Squared + Sin_D_C3_5term) *
                         Input_Squared + Sin_D_C1_5term);
         Inter_Result := Inter_Result * Real(Input);
         if Inter_Result > 1.0 then
            Inter_Result := 1.0;
         elsif Inter_Result < -1.0 then
            Inter_Result := -1.0;
         end if;
         Result := Sin_Cos_Ratio( Inter_Result );
         return Result;

      end Sin_D_5term;


      function Sin_D_4term(Input : Degrees) return Sin_Cos_Ratio is

         Input_Squared : Real;
         Inter_Result  : Real;
         Result        : Sin_Cos_Ratio;

      begin

         Input_Squared := Input * Input;
         Inter_Result   :=               (((Sin_D_C7_4term  *
                           Input_Squared + Sin_D_C5_4term) *
                           Input_Squared + Sin_D_C3_4term) *
                           Input_Squared + Sin_D_C1_4term);
         Inter_Result := Inter_Result * Real(Input);
         if Inter_Result > 1.0 then
            Inter_Result := 1.0;
         elsif Inter_Result < -1.0 then
```

```
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;

   end Sin_D_4term;


-- --cosine functions

   function Cos_D_5term(Input : Degrees) return Sin_Cos_Ratio is

      Input_Squared : Real;
      Inter_Result  : Real;
      Mod_Input     : Degrees;
      Result        : Sin_Cos_Ratio;

   begin

      Mod_Input := 90.0 - Input;
      Input_Squared := Mod_Input * Mod_Input;
      Inter_Result   :=              ((((Sin_D_C9_5term  *
                        Input_Squared + Sin_D_C7_5term) *
                        Input_Squared + Sin_D_C5_5term) *
                        Input_Squared + Sin_D_C3_5term) *
                        Input_Squared + Sin_D_C1_5term);
      Inter_Result := Inter_Result * Real(Mod_Input);
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;

   end Cos_D_5term;


   function Cos_D_4term(Input : Degrees) return Sin_Cos_Ratio is

      Input_Squared : Real;
      Inter_Result  : Real;
      Mod_Input     : Degrees;
      Result        : Sin_Cos_Ratio;

   begin

      Mod_Input := 90.0 - Input;
      Input_Squared := Mod_Input * Mod_Input;
      Inter_Result   :=                ((((Sin_D_C7_4term  *
                        Input_Squared + Sin_D_C5_4term) *
                        Input_Squared + Sin_D_C3_4term) *
                        Input_Squared + Sin_D_C1_4term);
      Inter_Result := Inter_Result * Real(Mod_Input);
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
```

```
            Inter_Result := -1.0;
         end if;
         Result := Sin_Cos_Ratio( Inter_Result );
         return Result;

      end Cos_D_4term;


-- -- Tangent function

      function Tan_D_5term (Input : Degrees) return Tan_Ratio is
         Sin : Sin_Cos_Ratio;
         Cos : Sin_Cos_Ratio;
      begin
         Sin := Sin_D_5term(Input);
         if Input < 0.0 then
            Cos := - Cos_D_5term( 180.0 + Input );
         else
            Cos := Cos_D_5term(Input);
         end if;
         return Tan_Ratio(Sin / Cos);
      end Tan_D_5term;

      function Tan_D_4term (Input : Degrees) return Tan_Ratio is
         Sin : Sin_Cos_Ratio;
         Cos : Sin_Cos_Ratio;
      begin
         Sin := Sin_D_4term(Input);
         if Input < 0.0 then
            Cos := - Cos_D_4term( 180.0 + Input );
         else
            Cos := Cos_D_4term(Input);
         end if;
         return Tan_Ratio(Sin / Cos);
      end Tan_D_4term;

end Hastings_Degree_Operations;
```

3.3.6.8.9.4.9.2.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.4.9.2.8 LIMITATIONS

None.

3.3.6.8.9.4.9.2.9 LLCSC DESIGN

None.

3.3.6.8.9.4.9.2.10 UNIT DESIGN

None.

3.3.6.8.9.4.10 UNIT DESIGN

None.

3.3.6.8.9.5 MODIFIED_NEWTON_RAPHSON PACKAGE DESIGN (CATALOG #P766-0)

This packages contains generic functions providing Modified Newton Raphson polynomial solutions for the square root function. Provisions are made for the square root functions to handle units of real. Outputs are also of type real.

The following table lists the catalog numbers for subunits contained in this part:

Name	Catalog _#
SqRt	P767-0

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R220.

3.3.6.8.9.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.5.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by the only function (Sqrt) in this part:

Name	Type	Description
Inputs	Floating point	Floating point Input to square root function.
Outputs	Floating point	Floating point Output of square root function.
Reals	Floating point	Floating point intermediate type to avoid constraint errors.

Data objects:

The following table describes the generic formal objects required by the only function (Sqrt) in this part:

Name	Type	Value	Description
Iteration_Num	Floating Point	Positive	Number of times the function performs the calculation cycle.

FORMAL PARAMETERS:

The following table describes the formal parameters for the functions contained in this part:

Function	Name	Type	Description
Sqrt	Input	Inputs	Input for square root function
	Iteration_No	Positive	Number of times to compute

3.3.6.8.9.5.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
C1	constant	2.18518_306	1st approximation constant
C2	constant	3.02289_917	2nd approximation constant
C3	constant	1.54515_776	3rd approximation constant

3.3.6.8.9.5.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.5.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials)
package body Modified_Newton_Raphson is

   C1 : constant :=  2.18518_306;
   C2 : constant :=  3.02289_917;
   C3 : constant :=  1.54515_776;


   function SqRt(Input : Inputs) return Outputs is

      Inter_Result : Reals;
      Result       : Outputs;
      Root_Pwr     : Reals;
```

```
      X_Norm       : Reals;

   begin
      if Input = 0.0 then
         Result := 0.0;
      else
         X_Norm   := Reals( Input );
--       ----------------------------------------
--       - Reduce input to between 0.25 and 1.0 -
--       - in order to achieve better initial   -
--       - approximation                        -
--       ----------------------------------------
         Root_Pwr := 1.0;
         if Input > 1.0 then
            Reduce:
               while X_Norm > 1.0 loop
                  Root_Pwr := Root_Pwr * 2.0;
                  X_Norm   := X_Norm * 0.25;
               end loop Reduce;
         else
            Increase:
               while X_Norm < 0.25 loop
                  Root_Pwr := Root_Pwr * 0.5;
                  X_Norm   := X_Norm * 4.0;
               end loop Increase;
         end if;
         Inter_Result := C1 - C2 / (X_Norm + C3);
         Inter_Result := (X_Norm/Inter_Result + Inter_Result) * 0.5;
         Inter_Result := (X_Norm/Inter_Result + Inter_Result) * 0.5;
         Inter_Result := (X_Norm/Inter_Result + Inter_Result) * 0.5;
         Inter_Result := Inter_Result * Root_Pwr;
         Result := Outputs(Inter_Result);
      end if;                    -- Input not 0.0
      return Result;
   end SqRt;

end Modified_Newton_Raphson;
```

3.3.6.8.9.5.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.5.8 LIMITATIONS

None.

3.3.6.8.9.5.9 LLCSC DESIGN

None.

3.3.6.8.9.5.10 UNIT DESIGN

None.

3.3.6.8.9.6 NEWTON_RAPHSON PACKAGE DESIGN (CATALOG #P768-0)

This packages contains generic functions providing Newton Raphson polynomial solutions for the square root function. Provisions are made for the square root functions to handle units of real. Outputs are also of type real.

The following table lists the catalog numbers for subunits contained in this part:

Name	Catalog _#
SqRt	P769-0

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R221.

3.3.6.8.9.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.6.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by the only function (Sqrt) in this part:

Name	Type	Description
Inputs	Floating point	Floating point Input to square root function.
Outputs	Floating point	Floating point Output of square root function.
Reals	Floating point	Floating point intermediate type to avoid constraint errors.

Data objects:

The following table describes the generic formal objects required by the only function (Sqrt) in this part:

Name	Type	Value	Description
Iteration_Num	Floating Point	Positive	Number of times the function performs the calculation cycle.

FORMAL PARAMETERS:

The following table describes the formal parameters for the functions contained in this part:

Function	Name	Type	Description
Sqrt	Input	Inputs	Input for square root function
	Iteration_No	Positive	Number of times to compute

3.3.6.8.9.6.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
C1	constant	3.57142_857	1st approximation constant
C2	constant	14.57725_95	2nd approximation constant
C3	constant	0.30612_2449	3rd approximation constant
C4	constant	4.79591_837	4th approximation constant
C4	constant	-0.16659_7251	4th approximation constant

3.3.6.8.9.6.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.6.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials)
package body Newton_Raphson is

   C1 : constant :=  3.57142_857;
   C2 : constant := 14.57725_95;
   C3 : constant :=  0.30612_2449;
   C4 : constant :=  4.79591_837;
   C5 : constant :=  0.16659_7251;

   function SqRt(Input : Inputs) return Outputs is
```

```
      Inter_Result : Reals;
      Result       : Outputs;
      Root_Pwr     : Reals;
      X_Norm       : Reals;

   begin
      if Input = 0.0 then
         Result := 0.0;
      else
         X_Norm   := Reals( Input );
--       ----------------------------------------
--       - Reduce input to between 0.25 and 1.0 -
--       - in order to achieve better initial   -
--       - approximation                        -
--       ----------------------------------------
         Root_Pwr := 1.0;
         if Input > 1.0 then
            Reduce:
               while X_Norm > 1.0 loop
                  Root_Pwr := Root_Pwr * 2.0;
                  X_Norm   := X_Norm * 0.25;
               end loop Reduce;
         else
            Increase:
               while X_Norm < 0.25 loop
                  Root_Pwr := Root_Pwr * 0.5;
                  X_Norm   := X_Norm * 4.0;
               end loop Increase;
         end if;
         Inter_Result := C1 - (C2 * (X_Norm + C3)) / ((X_Norm + C4) * (X_Norm + C5) + C5);
         Inter_Result := (X_Norm/Inter_Result + Inter_Result) * 0.5;
         Inter_Result := (X_Norm/Inter_Result + Inter_Result) * 0.5;
         Inter_Result := (X_Norm/Inter_Result + Inter_Result) * 0.5;
         Inter_Result := Inter_Result * Root_Pwr;
         Result := Outputs(Inter_Result);

      end if;
      return Result;
   end SqRt;

end Newton_Raphson;
```

3.3.6.8.9.6.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.6.8 LIMITATIONS

None.

3.3.6.8.9.6.9 LLCSC DESIGN

None.

3.3.6.8.9.6.10 UNIT DESIGN

None.

3.3.6.8.9.7 TAYLOR_SERIES PACKAGE DESIGN (CATALOG #P795-0)

This packages contains generic functions providing Taylor polynomial solutions for a set of trigonometric functions. Provisions are made for the trigonometric functions to handle units of radians or degrees.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.7.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R222.

3.3.6.8.9.7.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.7.3 INPUT/OUTPUT

None.

3.3.6.8.9.7.4 LOCAL DATA

None.

3.3.6.8.9.7.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.7.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials)
package body Taylor_Series is

   package body Taylor_Radian_Operations is separate;

   package body Taylor_Degree_Operations is separate;

   package body Taylor_Natural_Log is separate;

   package body Taylor_Log_Base_N is separate;

end Taylor_Series;
```

3.3.6.8.9.7.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.7.8 LIMITATIONS

None.

3.3.6.8.9.7.9 LLCSC DESIGN

3.3.6.8.9.7.9.1 TAYLOR_RADIAN_OPERATIONS PACKAGE DESIGN (CATALOG #P796-0)

This generic package contains functions providing Taylor polynomial solutions for a set of trigonometric functions. This package is designed to handle units of radians.

The following table lists the catalog numbers for subunits contained in this part:

Name	Catalog #
Sin_R_8term	P797-0
Sin_R_7term	P798-0
Sin_R_6term	P799-0
Sin_R_5term	P800-0
Sin_R_4term	P801-0
Cos_R_8term	P802-0
Cos_R_7term	P803-0
Cos_R_6term	P804-0
Cos_R_5term	P805-0
Cos_R_4term	P806-0
Tan_R_8term	P807-0
Arcsin_R_8term	P808-0
Arcsin_R_7term	P809-0
Arcsin_R_6term	P810-0
Arcsin_R_5term	P811-0
Arccos_R_8term	P812-0
Arccos_R_7term	P813-0
Arccos_R_6term	P814-0
Arccos_R_5term	P815-0
Arctan_R_8term	P816-0
Arctan_R_7term	P817-0
Arctan_R_6term	P818-0
Arctan_R_5term	P819-0
Arctan_R_4term	P820-0
Alt_Arctan_R_8term	P821-0
Alt_Arctan_R_7term	P822-0
Alt_Arctan_R_6term	P823-0
Alt_Arctan_R_5term	P824-0
Alt_Arctan_R_4term	P825-0
Mod_Sin_R_8term	P826-0
Mod_Sin_R_7term	P827-0
Mod_Sin_R_6term	P828-0
Mod_Sin_R_5term	P829-0
Mod_Sin_R_4term	P830-0
Mod_Cos_R_8term	P831-0
Mod_Cos_R_7term	P832-0
Mod_Cos_R_6term	P833-0
Mod_Cos_R_5term	P834-0
Mod_Cos_R_4term	P835-0
Mod_Tan_R_8term	P836-0
Mod_Tan_R_7term	P837-0
Mod_Tan_R_6term	P838-0
Mod_Tan_R_5term	P839-0
Mod_Tan_R_4term	P840-0

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.7.9.1.1 REQUIREMENTS ALLOCATION

This part partially meets CAMP requirement R222.

3.3.6.8.9.7.9.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.7.9.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Radians	Floating point	Allows floating point representation of radian measurements.
Real	Floating point	General floating point representation.
Sin_Cos_Ratio	Floating point	Represents sines and cosines.
Tan_Ratio	Floating point	Represents tangent values.

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
Pi_Over_2	Radians	constant	constant value Pi divided by 2
Pi_Over_4	Radians	constant	constant value Pi divided by 4

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Overloaded operator to multiply radians * radians yielding a real result.

FORMAL PARAMETERS:

The following table describes the formal parameters for the functions contained in this part:

Function	Name	Type	Description
Sine Functions	Input	Radians	Input for sine computation
Cosine Functions	Input	Radians	Input for cosine computation
Tangent Functions	Input	Radians	Input for tangent computation
Arcsine Functions	Input	Sin_Cos_Ratio	Input for Arcsine computation
Arccosine Functions	Input	Sin_Cos_Ratio	Input for Arccosine computation
Arctangent Functions	Input	Tan_Ratio	Input for Arctangent computation

3.3.6.8.9.7.9.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Sin_R_C3	constant	-0.16666_6666	3rd term coefficient
Sin_R_C5	constant	0.00833_33333_3	5th term coefficient
Sin_R_C7	constant	0.00019_84126_98	7th term coefficient
Sin_R_C9	constant	0.00000_27557_3164	9th term coefficient
Sin_R_C11	constant	0.00000_00250_51870_8	11th term coefficient
Sin_R_C13	constant	0.00000_00001_60478_446	13th term coefficient
Sin_R_C15	constant	0.00000_00000_00737_06627_7	15th term coefficient
Cos_R_C3	constant	-0.50000_0000	3rd term coefficient
Cos_R_C5	constant	0.04166_66666	5th term coefficient
Cos_R_C7	constant	-0.00138_88888_8	7th term coefficient
Cos_R_C9	constant	0.00002_48015_873	9th term coefficient
Cos_R_C11	constant	-0.00000_02755_73192	11th term coefficient
Cos_R_C13	constant	0.00000_00020_87675_69	13th term coefficient
Cos_R_C15	constant	-0.00000_00000_11470_7455	15th term coefficient
Tan_R_C3	constant	0.33333_3333	3rd term coefficient
Tan_R_C5	constant	0.13333_3333	5th term coefficient
Tan_R_C7	constant	0.05396_82539	7th term coefficient
Tan_R_C9	constant	0.02186_96649	9th term coefficient
Tan_R_C11	constant	0.00886_32355_2	11th term coefficient
Tan_R_C13	constant	0.00359_21280_3	13th term coefficient
Tan_R_C15	constant	0.00145_58343_8	15th term coefficient
Arcsin_R_C3	constant	0.16666_6666	3rd term coefficient
Arcsin_R_C5	constant	0.075	5th term coefficient
Arcsin_R_C7	constant	0.04464_28571	7th term coefficient
Arcsin_R_C9	constant	0.03038_19444	9th term coefficient
Arcsin_R_C11	constant	0.02237_21591	11th term coefficient
Arcsin_R_C13	constant	0.01735_27644	13th term coefficient

Arcsin_R_C15	constant	0.01396_48438	15th term coefficient
Arctan_R_C3	constant	0.33333_3333	3rd term coefficient
Arctan_R_C5	constant	-0.2	5th term coefficient
Arctan_R_C7	constant	0.14285_7114	7th term coefficient
Arctan_R_C9	constant	-0.11111_1111	9th term coefficient
Arctan_R_C11	constant	0.09090_90909	11th term coefficient
Arctan_R_C13	constant	-0.07692_30769	13th term coefficient
Arctan_R_C15	constant	0.06666_66667	15th term coefficient
Alt_Arctan_R_C3	constant	-0.33333_3333	3rd term coefficient
Alt_Arctan_R_C5	constant	0.2	5th term coefficient
Alt_Arctan_R_C7	constant	-0.14285_7114	7th term coefficient
Alt_Arctan_R_C9	constant	0.11111_1111	9th term coefficient
Alt_Arctan_R_C11	constant	-0.09090_90909	11th term coefficient
Alt_Arctan_R_C13	constant	0.07692_30769	13th term coefficient
Alt_Arctan_R_C15	constant	-0.06666_66667	15th term coefficient

3.3.6.8.9.7.9.1.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.7.9.1.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials.Taylor_Series)
package body Taylor_Radian_Operations is

-- -- The Sine constants are used in the Taylor operations for computing
   -- the sine.  The Cosine constants are used in computing cosines.  In
   -- the Modified Taylor operations, however, both sets of constants are
   -- used.  Constants are given for 9 digits of precision for both extended
   -- and single precision.  They are named to correspond to the power
   -- of X (Input) with which they are termed.

   Sin_R_C3  : constant := -0.16666_6667;
   Sin_R_C5  : constant :=  0.00833_33333_3;
   Sin_R_C7  : constant := -0.00019_84126_98;
   Sin_R_C9  : constant :=  0.00000_27557_3164;
   Sin_R_C11 : constant := -0.00000_00250_51870_9;
```

```
   Sin_R_C13 : constant :=  0.00000_00001_60478_446;
   Sin_R_C15 : constant := -0.00000_00000_00737_06627_8;

   Cos_R_C3  : constant := -0.50000_0000;
   Cos_R_C5  : constant :=  0.04166_66667;
   Cos_R_C7  : constant := -0.00138_88888_9;
   Cos_R_C9  : constant :=  0.00002_48015_873;
   Cos_R_C11 : constant := -0.00000_02755_73192;
   Cos_R_C13 : constant :=  0.00000_00020_87675_70;
   Cos_R_C15 : constant := -0.00000_00000_11470_7456;

   Tan_R_C3  : constant := 0.33333_3333;
   Tan_R_C5  : constant := 0.13333_3333;
   Tan_R_C7  : constant := 0.05396_82540;
   Tan_R_C9  : constant := 0.02186_9488;
   Tan_R_C11 : constant := 0.00886_32355_3;
   Tan_R_C13 : constant := 0.00359_21280_4;
   Tan_R_C15 : constant := 0.00145_58343_9;

   Arcsin_R_C3  : constant := 0.16666_6666;
   Arcsin_R_C5  : constant := 0.075;
   Arcsin_R_C7  : constant := 0.04464_28571;
   Arcsin_R_C9  : constant := 0.03038_19444;
   Arcsin_R_C11 : constant := 0.02237_21591;
   Arcsin_R_C13 : constant := 0.01735_27644;
   Arcsin_R_C15 : constant := 0.01396_48438;

   Arctan_R_C3  : constant :=  0.33333_3333;
   Arctan_R_C5  : constant := -0.2;
   Arctan_R_C7  : constant :=  0.14285_7142;
   Arctan_R_C9  : constant := -0.11111_1111;
   Arctan_R_C11 : constant :=  0.09090_90909;
   Arctan_R_C13 : constant := -0.07692_30769;
   Arctan_R_C15 : constant :=  0.06666_66667;

   Alt_Arctan_R_C3  : constant := -0.33333_3333;
   Alt_Arctan_R_C5  : constant :=  0.2;
   Alt_Arctan_R_C7  : constant := -0.14285_7142;
   Alt_Arctan_R_C9  : constant :=  0.11111_1111;
   Alt_Arctan_R_C11 : constant := -0.09090_90909;
   Alt_Arctan_R_C13 : constant :=  0.07692_30769;
   Alt_Arctan_R_C15 : constant := -0.06666_66667;


-- -- Taylor Sine functions

   function Sin_R_8term (Input : Radians) return Sin_Cos_Ratio is

      Inter_result  : Real;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Radians;

   begin
      X_Squared := Input * Input;
      Inter_Result :=  ((((((Sin_R_C15  * X_Squared +
                             Sin_R_C13) * X_Squared +
                             Sin_R_C11) * X_Squared +
```

```
                                Sin_R_C9)  * X_Squared +
                                Sin_R_C7)  * X_Squared +
                                Sin_R_C5)  * X_Squared +
                                Sin_R_C3)  * X_Squared;
      Inter_Result := Inter_Result * Real(Input) + Real(Input);
      If Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Sin_R_8term;


   function Sin_R_7term (Input : Radians) return Sin_Cos_Ratio is

      Inter_result  : Real;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Radians;

   begin
      X_Squared := Input * Input;
      Inter_Result :=  (((((Sin_R_C13  * X_Squared +
                            Sin_R_C11) * X_Squared +
                            Sin_R_C9)  * X_Squared +
                            Sin_R_C7)  * X_Squared +
                            Sin_R_C5)  * X_Squared +
                            Sin_R_C3)  * X_Squared;
      Inter_Result := Inter_Result * Real(Input) + Real(Input);
      If Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Sin_R_7term;


   function Sin_R_6term (Input : Radians) return Sin_Cos_Ratio is

      Inter_result  : Real;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Radians;

   begin
      X_Squared := Input * Input;
      Inter_Result :=  ((((Sin_R_C11 * X_Squared +
                           Sin_R_C9) * X_Squared +
                           Sin_R_C7) * X_Squared +
                           Sin_R_C5) * X_Squared +
                           Sin_R_C3) * X_Squared;
      Inter_Result := Inter_Result * Real(Input) + Real(Input);
      If Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
```

```
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Sin_R_6term;


   function Sin_R_5term (Input : Radians) return Sin_Cos_Ratio is

      Inter_result  : Real;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Radians;

   begin
      X_Squared := Input * Input;
      Inter_Result :=  (((Sin_R_C9  * X_Squared +
                          Sin_R_C7) * X_Squared +
                          Sin_R_C5) * X_Squared +
                          Sin_R_C3) * X_Squared;
      Inter_Result := Inter_Result * Real(Input) + Real(Input);
      If Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Sin_R_5term;


   function Sin_R_4term (Input : Radians) return Sin_Cos_Ratio is

      Inter_result  : Real;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Radians;

   begin
      X_Squared := Input * Input;
      Inter_Result :=  ((Sin_R_C7  * X_Squared +
                         Sin_R_C5) * X_Squared +
                         Sin_R_C3) * X_Squared;
      Inter_Result := Inter_Result * Real(Input) + Real(Input);
      If Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Sin_R_4term;


-- -- Taylor Cosine functions

   function Cos_R_8term (Input : Radians) return Sin_Cos_Ratio is

      Inter_result  : Real;
```

```
      Mod_Input      : Radians;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Radians;

   begin
      if Input > Pi_Over_2 then
         Mod_Input := Pi - Input;
      else
         Mod_Input := Input;
      end if;
      X_Squared := Mod_Input * Mod_Input;
      Inter_Result := ((((((Cos_R_C15  * X_Squared +
                            Cos_R_C13) * X_Squared +
                            Cos_R_C11) * X_Squared +
                            Cos_R_C9) * X_Squared +
                            Cos_R_C7) * X_Squared +
                            Cos_R_C5) * X_Squared +
                            Cos_R_C3) * X_Squared;
      Inter_Result := Inter_Result + 1.0;
      if Input > Pi_Over_2 then
         Inter_Result := - Inter_Result;
      end if;
      If Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
   return Result;
   end Cos_R_8term;


   function Cos_R_7term (Input : Radians) return Sin_Cos_Ratio is

      Inter_result   : Real;
      Mod_Input      : Radians;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Radians;

   begin
      If Input > Pi_Over_2 then
         Mod_Input := Pi - Input;
      else
         Mod_Input := Input;
      end if;
      X_Squared := Mod_Input * Mod_Input;
      Inter_Result := (((((Cos_R_C13  * X_Squared +
                           Cos_R_C11) * X_Squared +
                           Cos_R_C9) * X_Squared +
                           Cos_R_C7) * X_Squared +
                           Cos_R_C5) * X_Squared +
                           Cos_R_C3) * X_Squared;
      Inter_Result := Inter_Result + 1.0;
      If Input > Pi_Over_2 then
         Inter_Result := - Inter_Result;
      end if;
      If Inter_Result > 1.0 then
```

```
      Inter_Result := 1.0;
   elsif Inter_Result < -1.0 then
      Inter_Result := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result );
return Result;
end Cos_R_7term;


function Cos_R_6term (Input : Radians) return Sin_Cos_Ratio is

   Inter_result  : Real;
   Mod_Input     : Radians;
   Result        : Sin_Cos_Ratio;
   X_Squared     : Radians;

begin
   If Input > Pi_Over_2 then
      Mod_Input := Pi - Input;
   else
      Mod_Input := Input;
   end if;
   X_Squared := Mod_Input * Mod_Input;
   Inter_Result :=  ((((Cos_R_C11  * X_Squared +
                       Cos_R_C9) * X_Squared +
                       Cos_R_C7) * X_Squared +
                       Cos_R_C5) * X_Squared +
                       Cos_R_C3) * X_Squared;
   Inter_Result := Inter_Result + 1.0;
   If Input > Pi_Over_2 then
      Inter_Result := - Inter_Result;
   end if;
   If Inter_Result > 1.0 then
      Inter_Result := 1.0;
   elsif Inter_Result < -1.0 then
      Inter_Result := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result );
return Result;
end Cos_R_6term;


function Cos_R_5term (Input : Radians) return Sin_Cos_Ratio is

   Inter_result  : Real;
   Mod_Input     : Radians;
   Result        : Sin_Cos_Ratio;
   X_Squared     : Radians;

begin
   If Input > Pi_Over_2 then
      Mod_Input := Pi - Input;
   else
      Mod_Input := Input;
   end if;
   X_Squared := Mod_Input * Mod_Input;
   Inter_Result :=  (((Cos_R_C9  * X_Squared +
```

```
                        Cos_R_C7) * X_Squared +
                        Cos_R_C5) * X_Squared +
                        Cos_R_C3) * X_Squared;
      Inter_Result := Inter_Result + 1.0;
      If Input > Pi_Over_2 then
         Inter_Result := - Inter_Result;
      end if;
      If Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
   return Result;
   end Cos_R_5term;


   function Cos_R_4term (Input : Radians) return Sin_Cos_Ratio is

      Inter_result  : Real;
      Mod_Input     : Radians;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Radians;

   begin
      If Input > Pi_Over_2 then
         Mod_Input := Pi - Input;
      else
         Mod_Input := Input;
      end if;
      X_Squared := Mod_Input * Mod_Input;
      Inter_Result :=  ((Cos_R_C7  * X_Squared +
                         Cos_R_C5) * X_Squared +
                         Cos_R_C3) * X_Squared;
      Inter_Result := Inter_Result + 1.0;
      If Input > Pi_Over_2 then
         Inter_Result := - Inter_Result;
      end if;
      If Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
   return Result;
   end Cos_R_4term;


-- -- Taylor tangent functions

   function Tan_R_8term (Input : Radians) return Tan_Ratio is

      Inter_result  : Real;
      Result        : Tan_Ratio;
      X_Squared     : Radians;

   begin
```

```
      X_Squared := Input * Input;
      Inter_Result :=  ((((((Tan_R_C15  * X_Squared +
                             Tan_R_C13) * X_Squared +
                             Tan_R_C11) * X_Squared +
                             Tan_R_C9)  * X_Squared +
                             Tan_R_C7)  * X_Squared +
                             Tan_R_C5)  * X_Squared +
                             Tan_R_C3)  * X_Squared;
      Result := Tan_Ratio(Inter_Result * Real(Input) + Real(Input));
      return Result;
   end Tan_R_8term;


-- -- Taylor arcsine functions

   function Arcsin_R_8term (Input : Sin_Cos_Ratio) return Radians is

      Inter_result  : Real;
      Result        : Radians;
      X_Squared     : Real;

   begin
      X_Squared := Real(Input * Input);
      Inter_Result :=  ((((((Arcsin_R_C15  * X_Squared +
                             Arcsin_R_C13) * X_Squared +
                             Arcsin_R_C11) * X_Squared +
                             Arcsin_R_C9)  * X_Squared +
                             Arcsin_R_C7)  * X_Squared +
                             Arcsin_R_C5)  * X_Squared +
                             Arcsin_R_C3)  * X_Squared;
      Result := Radians(Inter_Result * Real(Input) + Real(Input));
      return Result;
   end Arcsin_R_8term;


   function Arcsin_R_7term (Input : Sin_Cos_Ratio) return Radians is

      Inter_result  : Real;
      Result        : Radians;
      X_Squared     : Real;

   begin
      X_Squared := Real(Input * Input);
      Inter_Result :=  (((((Arcsin_R_C13  * X_Squared +
                           Arcsin_R_C11) * X_Squared +
                           Arcsin_R_C9)  * X_Squared +
                           Arcsin_R_C7)  * X_Squared +
                           Arcsin_R_C5)  * X_Squared +
                           Arcsin_R_C3)  * X_Squared;
      Result := Radians(Inter_Result * Real(Input) + Real(Input));
      return Result;
   end Arcsin_R_7term;


   function Arcsin_R_6term (Input : Sin_Cos_Ratio) return Radians is

      Inter_result  : Real;
```

```
      Result        : Radians;
      X_Squared     : Real;

   begin
      X_Squared := Real(Input * Input);
      Inter_Result :=  ((((Arcsin_R_C11  * X_Squared +
                           Arcsin_R_C9)  * X_Squared +
                           Arcsin_R_C7)  * X_Squared +
                           Arcsin_R_C5)  * X_Squared +
                           Arcsin_R_C3)  * X_Squared;
      Result := Radians(Inter_Result * Real(Input) + Real(Input));
      return Result;
   end Arcsin_R_6term;


   function Arcsin_R_5term (Input : Sin_Cos_Ratio) return Radians is

      Inter_result  : Real;
      Result        : Radians;
      X_Squared     : Real;

   begin
      X_Squared := Real(Input * Input);
      Inter_Result :=  (((Arcsin_R_C9  * X_Squared +
                          Arcsin_R_C7) * X_Squared +
                          Arcsin_R_C5) * X_Squared +
                          Arcsin_R_C3) * X_Squared;
      Result := Radians(Inter_Result * Real(Input) + Real(Input));
      return Result;
   end Arcsin_R_5term;


-- -- Taylor arccosine functions

   function Arccos_R_8term (Input : Sin_Cos_Ratio) return Radians is

      Inter_result  : Real;
      Result        : Radians;
      X_Squared     : Real;

   begin
      X_Squared := Real(Input * Input);
      Inter_Result :=  ((((((Arcsin_R_C15  * X_Squared +
                             Arcsin_R_C13) * X_Squared +
                             Arcsin_R_C11) * X_Squared +
                             Arcsin_R_C9)  * X_Squared +
                             Arcsin_R_C7)  * X_Squared +
                             Arcsin_R_C5)  * X_Squared +
                             Arcsin_R_C3)  * X_Squared;
      Result := Pi_Over_2 - (Radians(Inter_Result * Real(Input) + Real(Input)));
      return Result;
   end Arccos_R_8term;


   function Arccos_R_7term (Input : Sin_Cos_Ratio) return Radians is

      Inter_result  : Real;
```

```
      Result        : Radians;
      X_Squared     : Real;

   begin
      X_Squared := Real(Input * Input);
      Inter_Result :=  (((((Arcsin_R_C13  * X_Squared +
                            Arcsin_R_C11) * X_Squared +
                            Arcsin_R_C9)  * X_Squared +
                            Arcsin_R_C7)  * X_Squared +
                            Arcsin_R_C5)  * X_Squared +
                            Arcsin_R_C3)  * X_Squared;
      Result := Pi_Over_2 - (Radians(Inter_Result * Real(Input) + Real(Input)));
      return Result;
   end Arccos_R_7term;


   function Arccos_R_6term (Input : Sin_Cos_Ratio) return Radians is

      Inter_result  : Real;
      Result        : Radians;
      X_Squared     : Real;

   begin
      X_Squared := Real(Input * Input);
      Inter_Result :=  ((((Arcsin_R_C11  * X_Squared +
                           Arcsin_R_C9)  * X_Squared +
                           Arcsin_R_C7)  * X_Squared +
                           Arcsin_R_C5)  * X_Squared +
                           Arcsin_R_C3)  * X_Squared;
      Result := Pi_Over_2 - (Radians(Inter_Result * Real(Input) + Real(Input)));
      return Result;
   end Arccos_R_6term;


   function Arccos_R_5term (Input : Sin_Cos_Ratio) return Radians is

      Inter_result  : Real;
      Result        : Radians;
      X_Squared     : Real;

   begin
      X_Squared := Real(Input * Input);
      Inter_Result :=  (((Arcsin_R_C9  * X_Squared +
                          Arcsin_R_C7) * X_Squared +
                          Arcsin_R_C5) * X_Squared +
                          Arcsin_R_C3) * X_Squared;
      Result := Pi_Over_2 - (Radians(Inter_Result * Real(Input) + Real(Input)));
      return Result;
   end Arccos_R_5term;


-- -- Taylor Arctangent functions
-- -- Used when |Input| > 1

   function Arctan_R_8term(Input : Tan_Ratio) return Radians is

      Input_Squared : Radians;
```

```
      Inverse       : Tan_Ratio;
      Result        : Radians;
      Temp          : Radians;

   begin
      if Input <= 1.0 then
         Temp := -Pi_Over_2;
      else
         Temp := Pi_Over_2;
      end if;
      Inverse := 1.0 / Input;
      Input_Squared := Radians(Inverse * Inverse);
      Result := Temp + (((((((Arctan_R_C15  * Input_Squared +
                              Arctan_R_C13) * Input_Squared +
                              Arctan_R_C11) * Input_Squared +
                              Arctan_R_C9)  * Input_Squared +
                              Arctan_R_C7)  * Input_Squared +
                              Arctan_R_C5)  * Input_Squared +
                              Arctan_R_C3)  * Input_Squared -
                              1.0) * Radians(Inverse);
      return Result;
   end Arctan_R_8term;


   function Arctan_R_7term(Input : Tan_Ratio) return Radians is

      Input_Squared : Radians;
      Inverse       : Tan_Ratio;
      Result        : Radians;
      Temp          : Radians;

   begin
      if Input <= 1.0 then
         Temp := -Pi_Over_2;
      else
         Temp := Pi_Over_2;
      end if;
      Inverse := 1.0 / Input;
      Input_Squared := Radians(Inverse * Inverse);
      Result := Temp + ((((((Arctan_R_C13  * Input_Squared +
                             Arctan_R_C11) * Input_Squared +
                             Arctan_R_C9)  * Input_Squared +
                             Arctan_R_C7)  * Input_Squared +
                             Arctan_R_C5)  * Input_Squared +
                             Arctan_R_C3)  * Input_Squared -
                             1.0) * Radians(Inverse);
      return Result;
   end Arctan_R_7term;


   function Arctan_R_6term(Input : Tan_Ratio) return Radians is

      Input_Squared : Radians;
      Inverse       : Tan_Ratio;
      Result        : Radians;
      Temp          : Radians;
```

```
   begin
      if Input <= 1.0 then
         Temp := -Pi_Over_2;
      else
         Temp := Pi_Over_2;
      end if;
      Inverse := 1.0 / Input;
      Input_Squared := Radians(Inverse * Inverse);
      Result := Temp + (((((Arctan_R_C11 * Input_Squared +
                            Arctan_R_C9) * Input_Squared +
                            Arctan_R_C7) * Input_Squared +
                            Arctan_R_C5) * Input_Squared +
                            Arctan_R_C3) * Input_Squared -
                            1.0) * Radians(Inverse);
      return Result;
   end Arctan_R_6term;


   function Arctan_R_5term(Input : Tan_Ratio) return Radians is

      Input_Squared : Radians;
      Inverse       : Tan_Ratio;
      Result        : Radians;
      Temp          : Radians;

   begin
      if Input <= 1.0 then
         Temp := -Pi_Over_2;
      else
         Temp := Pi_Over_2;
      end if;
      Inverse := 1.0 / Input;
      Input_Squared := Radians(Inverse * Inverse);
      Result := Temp + ((((Arctan_R_C9 * Input_Squared +
                           Arctan_R_C7) * Input_Squared +
                           Arctan_R_C5) * Input_Squared +
                           Arctan_R_C3) * Input_Squared -
                           1.0) * Radians(Inverse);
      return Result;
   end Arctan_R_5term;


   function Arctan_R_4term(Input : Tan_Ratio) return Radians is

      Input_Squared : Radians;
      Inverse       : Tan_Ratio;
      Result        : Radians;
      Temp          : Radians;

   begin
      if Input <= 1.0 then
         Temp := -Pi_Over_2;
      else
         Temp := Pi_Over_2;
      end if;
      Inverse := 1.0 / Input;
      Input_Squared := Radians(Inverse * Inverse);
```

```
      Result := Temp + (((Arctan_R_C7  * Input_Squared +
                          Arctan_R_C5) * Input_Squared +
                          Arctan_R_C3) * Input_Squared -
                          1.0) * Radians(Inverse);
      return Result;
   end Arctan_R_4term;


-- -- Alternate Taylor Arctangent functions
-- -- Used when |Input| < 1

   function Alt_Arctan_R_8term(Input : Tan_Ratio) return Radians is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Result        : Radians;

   begin
      Input_Squared := Input * Input;
      Inter_Result := ((((((Alt_Arctan_R_C15  * Input_Squared +
                          Alt_Arctan_R_C13) * Input_Squared +
                          Alt_Arctan_R_C11) * Input_Squared +
                          Alt_Arctan_R_C9)  * Input_Squared +
                          Alt_Arctan_R_C7)  * Input_Squared +
                          Alt_Arctan_R_C5)  * Input_Squared +
                          Alt_Arctan_R_C3)  * Input_Squared;
      Result := Radians(Inter_Result * Input + Input);
      return Result;
   end Alt_Arctan_R_8term;


   function Alt_Arctan_R_7term(Input : Tan_Ratio) return Radians is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Result        : Radians;

   begin
      Input_Squared := Input * Input;
      Inter_Result := (((((Alt_Arctan_R_C13  * Input_Squared +
                          Alt_Arctan_R_C11) * Input_Squared +
                          Alt_Arctan_R_C9)  * Input_Squared +
                          Alt_Arctan_R_C7)  * Input_Squared +
                          Alt_Arctan_R_C5)  * Input_Squared +
                          Alt_Arctan_R_C3)  * Input_Squared;
      Result := Radians(Inter_Result * Input + Input);
      return Result;
   end Alt_Arctan_R_7term;


   function Alt_Arctan_R_6term(Input : Tan_Ratio) return Radians is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Result        : Radians;

   begin
```

```
      Input_Squared := Input * Input;
      Inter_Result := ((((Alt_Arctan_R_C11 * Input_Squared +
                          Alt_Arctan_R_C9) * Input_Squared +
                          Alt_Arctan_R_C7) * Input_Squared +
                          Alt_Arctan_R_C5) * Input_Squared +
                          Alt_Arctan_R_C3) * Input_Squared;
      Result := Radians(Inter_Result * Input + Input);
      return Result;
   end Alt_Arctan_R_6term;


   function Alt_Arctan_R_5term(Input : Tan_Ratio) return Radians is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Result        : Radians;

   begin
      Input_Squared := Input * Input;
      Inter_Result := (((Alt_Arctan_R_C9  * Input_Squared +
                         Alt_Arctan_R_C7) * Input_Squared +
                         Alt_Arctan_R_C5) * Input_Squared +
                         Alt_Arctan_R_C3) * Input_Squared;
      Result := Radians(Inter_Result * Input + Input);
      return Result;
   end Alt_Arctan_R_5term;


   function Alt_Arctan_R_4term(Input : Tan_Ratio) return Radians is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Result        : Radians;

   begin
      Input_Squared := Input * Input;
      Inter_Result := ((Alt_Arctan_R_C7  * Input_Squared +
                        Alt_Arctan_R_C5) * Input_Squared +
                        Alt_Arctan_R_C3) * Input_Squared;
      Result := Radians(Inter_Result * Input + Input);
      return Result;
   end Alt_Arctan_R_4term;


-- -- Modified Taylor Sine functions

   function Mod_Sin_R_8term(Input : Radians) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if abs(Input) >= Pi_Over_4 then
         Inter_Result_0 := Real(Pi_Over_2 - Abs(Input));
         X_Squared := Inter_Result_0 * Inter_Result_0;
```

```
      Inter_Result_1 := ((((((Cos_R_C15  * X_Squared +
                              Cos_R_C13) * X_Squared +
                              Cos_R_C11) * X_Squared +
                              Cos_R_C9)  * X_Squared +
                              Cos_R_C7)  * X_Squared +
                              Cos_R_C5)  * X_Squared +
                              Cos_R_C3)  * X_Squared;
      Inter_Result_1 := Inter_Result_1 + 1.0;
      If Input <= - Pi_Over_4 then
         Inter_Result_1 := - Inter_Result_1;
      end if;
   else
      X_Squared := Input * Input;
      Inter_Result_1 := ((((((Sin_R_C15  * X_Squared +
                              Sin_R_C13) * X_Squared +
                              Sin_R_C11) * X_Squared +
                              Sin_R_C9)  * X_Squared +
                              Sin_R_C7)  * X_Squared +
                              Sin_R_C5)  * X_Squared +
                              Sin_R_C3)  * X_Squared;
      Inter_Result_1 := Inter_Result_1 * Real(Input) + Real(Input);
   end if;
   if Inter_Result_1 > 1.0 then
      Inter_Result_1 := 1.0;
   elsif Inter_Result_1 < -1.0 then
      Inter_Result_1 := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result_1 );
   return Result;
end Mod_Sin_R_8term;


function Mod_Sin_R_7term (Input : Radians) return Sin_Cos_Ratio is

   Inter_Result_0 : Real;
   Inter_Result_1 : Real;
   Result         : Sin_Cos_Ratio;
   X_Squared      : Real;

begin
   if abs(Input) >= Pi_Over_4 then
      Inter_Result_0 := Real(Pi_Over_2 - Abs(Input));
      X_Squared := Inter_Result_0 * Inter_Result_0;
      Inter_Result_1 := (((((Cos_R_C13  * X_Squared +
                              Cos_R_C11) * X_Squared +
                              Cos_R_C9)  * X_Squared +
                              Cos_R_C7)  * X_Squared +
                              Cos_R_C5)  * X_Squared +
                              Cos_R_C3)  * X_Squared;
      Inter_Result_1 := Inter_Result_1 + 1.0;
      If Input <= - Pi_Over_4 then
         Inter_Result_1 := - Inter_Result_1;
      end if;
   else
      X_Squared := Input * Input;
      Inter_Result_1 := (((((Sin_R_C13  * X_Squared +
                              Sin_R_C11) * X_Squared +
```

```
                                Sin_R_C9)  * X_Squared +
                                Sin_R_C7)  * X_Squared +
                                Sin_R_C5)  * X_Squared +
                                Sin_R_C3)  * X_Squared;
      Inter_Result_1 := Inter_Result_1 * Real(Input) + Real(Input);
   end if;
   if Inter_Result_1 > 1.0 then
      Inter_Result_1 := 1.0;
   elsif Inter_Result_1 < -1.0 then
      Inter_Result_1 := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result_1 );
   return Result;
end Mod_Sin_R_7term;


function Mod_Sin_R_6term (Input : Radians) return Sin_Cos_Ratio is

   Inter_Result_0 : Real;
   Inter_Result_1 : Real;
   Result         : Sin_Cos_Ratio;
   X_Squared      : Real;

begin
   if abs(Input) >= Pi_Over_4 then
      Inter_Result_0 := Real(Pi_Over_2 - Abs(Input));
      X_Squared := Inter_Result_0 * Inter_Result_0;
      Inter_Result_1 := ((((Cos_R_C11  * X_Squared +
                            Cos_R_C9) * X_Squared +
                            Cos_R_C7) * X_Squared +
                            Cos_R_C5) * X_Squared +
                            Cos_R_C3) * X_Squared;
      Inter_Result_1 := Inter_Result_1 + 1.0;
      If Input <= - Pi_Over_4 then
         Inter_Result_1 := - Inter_Result_1;
      end if;
   else
      X_Squared := Input * Input;
      Inter_Result_1 := ((((Sin_R_C11  * X_Squared +
                            Sin_R_C9) * X_Squared +
                            Sin_R_C7) * X_Squared +
                            Sin_R_C5) * X_Squared +
                            Sin_R_C3) * X_Squared;
      Inter_Result_1 := Inter_Result_1 * Real(Input) + Real(Input);
   end if;
   if Inter_Result_1 > 1.0 then
      Inter_Result_1 := 1.0;
   elsif Inter_Result_1 < -1.0 then
      Inter_Result_1 := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result_1 );
   return Result;
end Mod_Sin_R_6term;


function Mod_Sin_R_5term (Input : Radians) return Sin_Cos_Ratio is
```

```
      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if abs(Input) >= Pi_Over_4 then
         Inter_Result_0 := Real(Pi_Over_2 - Abs(Input));
         X_Squared := Inter_Result_0 * Inter_Result_0;
         Inter_Result_1 := (((Cos_R_C9  * X_Squared +
                              Cos_R_C7) * X_Squared +
                              Cos_R_C5) * X_Squared +
                              Cos_R_C3) * X_Squared;
         Inter_Result_1 := Inter_Result_1 + 1.0;
         If Input <= - Pi_Over_4 then
            Inter_Result_1 := - Inter_Result_1;
         end if;
      else
         X_Squared := Input * Input;
         Inter_Result_1 := (((Sin_R_C9  * X_Squared +
                              Sin_R_C7) * X_Squared +
                              Sin_R_C5) * X_Squared +
                              Sin_R_C3) * X_Squared;
         Inter_Result_1 := Inter_Result_1 * Real(Input) + Real(Input);
      end if;
      if Inter_Result_1 > 1.0 then
         Inter_Result_1 := 1.0;
      elsif Inter_Result_1 < -1.0 then
         Inter_Result_1 := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result_1 );
      return Result;
   end Mod_Sin_R_5term;


   function Mod_Sin_R_4term (Input : Radians) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if abs(Input) >= Pi_Over_4 then
         Inter_Result_0 := Real(Pi_Over_2 - Abs(Input));
         X_Squared := Inter_Result_0 * Inter_Result_0;
         Inter_Result_1 := ((Cos_R_C7  * X_Squared +
                             Cos_R_C5) * X_Squared +
                             Cos_R_C3) * X_Squared;
         Inter_Result_1 := Inter_Result_1 + 1.0;
         If Input <= - Pi_Over_4 then
            Inter_Result_1 := - Inter_Result_1;
         end if;
      else
         X_Squared := Input * Input;
         Inter_Result_1 := ((Sin_R_C7  * X_Squared +
                             Sin_R_C5) * X_Squared +
```

```
                              Sin_R_C3) * X_Squared;
      Inter_Result_1 := Inter_Result_1 * Real(Input) + Real(Input);
   end if;
   if Inter_Result_1 > 1.0 then
      Inter_Result_1 := 1.0;
   elsif Inter_Result_1 < -1.0 then
      Inter_Result_1 := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result_1 );
   return Result;
end Mod_Sin_R_4term;


-- -- Modified Taylor Cosine functions

   function Mod_Cos_R_8term(Input : Radians) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Mod_Input      : Radians;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if (Input <= Pi_Over_4) or (Input >= 3.0 * Pi_Over_4) then
         if Input > Pi_Over_2 then
            Mod_Input := Pi - Input;
         else
            Mod_Input := Input;
         end if;
         X_Squared := Mod_Input * Mod_Input;
         Inter_Result_1 := ((((((Cos_R_C15  * X_Squared +
                                 Cos_R_C13) * X_Squared +
                                 Cos_R_C11) * X_Squared +
                                 Cos_R_C9) * X_Squared +
                                 Cos_R_C7) * X_Squared +
                                 Cos_R_C5) * X_Squared +
                                 Cos_R_C3) * X_Squared;
         Inter_Result_1 :=  Inter_Result_1 + 1.0 ;
         If Input > Pi_Over_2 then
            Inter_Result_1 := - Inter_Result_1;
         end if;
      else
         Inter_Result_0 := Real(Pi_Over_2 - Input);
         X_Squared := Inter_Result_0 * Inter_Result_0;
         Inter_Result_1 := ((((((Sin_R_C15  * X_Squared +
                                 Sin_R_C13) * X_Squared +
                                 Sin_R_C11) * X_Squared +
                                 Sin_R_C9) * X_Squared +
                                 Sin_R_C7) * X_Squared +
                                 Sin_R_C5) * X_Squared +
                                 Sin_R_C3) * X_Squared;
         Inter_Result_1 := Inter_Result_1 * Inter_Result_0 + Inter_Result_0;
      end if;
      If Inter_Result_1 > 1.0 then
         Inter_Result_1 := 1.0;
      elsif Inter_Result_1 < -1.0 then
```

```
         Inter_Result_1 := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result_1 );
      return Result;
   end Mod_Cos_R_8term;


   function Mod_Cos_R_7term(Input : Radians) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Mod_Input      : Radians;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if (Input <= Pi_Over_4) or (Input >= 3.0 * Pi_Over_4) then
         if Input > Pi_Over_2 then
            Mod_Input := Pi - Input;
         else
            Mod_Input := Input;
         end if;
         X_Squared := Mod_Input * Mod_Input;
         Inter_Result_1 := (((((Cos_R_C13  * X_Squared +
                                Cos_R_C11) * X_Squared +
                                Cos_R_C9) * X_Squared +
                                Cos_R_C7) * X_Squared +
                                Cos_R_C5) * X_Squared +
                                Cos_R_C3) * X_Squared;
         Inter_Result_1 :=  Inter_Result_1 + 1.0 ;
         If Input > Pi_Over_2 then
            Inter_Result_1 := - Inter_Result_1;
         end if;
      else
         Inter_Result_0 := Real(Pi_Over_2 - Input);
         X_Squared := Inter_Result_0 * Inter_Result_0;
         Inter_Result_1 := (((((Sin_R_C13  * X_Squared +
                                Sin_R_C11) * X_Squared +
                                Sin_R_C9) * X_Squared +
                                Sin_R_C7) * X_Squared +
                                Sin_R_C5) * X_Squared +
                                Sin_R_C3) * X_Squared;
         Inter_Result_1 := Inter_Result_1 * Inter_Result_0 + Inter_Result_0;
      end if;
      If Inter_Result_1 > 1.0 then
         Inter_Result_1 := 1.0;
      elsif Inter_Result_1 < -1.0 then
         Inter_Result_1 := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result_1 );
      return Result;
   end Mod_Cos_R_7term;


   function Mod_Cos_R_6term(Input : Radians) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
```

```
      Inter_Result_1 : Real;
      Mod_Input      : Radians;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if (Input <= Pi_Over_4) or (Input >= 3.0 * Pi_Over_4) then
         if Input > Pi_Over_2 then
            Mod_Input := Pi - Input;
         else
            Mod_Input := Input;
         end if;
         X_Squared := Mod_Input * Mod_Input;
         Inter_Result_1 := ((((Cos_R_C11  * X_Squared +
                               Cos_R_C9) * X_Squared +
                               Cos_R_C7) * X_Squared +
                               Cos_R_C5) * X_Squared +
                               Cos_R_C3) * X_Squared;
         Inter_Result_1 :=  Inter_Result_1 + 1.0 ;
         If Input > Pi_Over_2 then
            Inter_Result_1 := - Inter_Result_1;
         end if;
      else
         Inter_Result_0 := Real(Pi_Over_2 - Input);
         X_Squared := Inter_Result_0 * Inter_Result_0;
         Inter_Result_1 := ((((Sin_R_C11  * X_Squared +
                               Sin_R_C9) * X_Squared +
                               Sin_R_C7) * X_Squared +
                               Sin_R_C5) * X_Squared +
                               Sin_R_C3) * X_Squared;
         Inter_Result_1 := Inter_Result_1 * Inter_Result_0 + Inter_Result_0;
      end if;
      If Inter_Result_1 > 1.0 then
         Inter_Result_1 := 1.0;
      elsif Inter_Result_1 < -1.0 then
         Inter_Result_1 := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result_1 );
      return Result;
   end Mod_Cos_R_6term;


   function Mod_Cos_R_5term(Input : Radians) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Mod_Input      : Radians;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if (Input <= Pi_Over_4) or (Input >= 3.0 * Pi_Over_4) then
         if Input > Pi_Over_2 then
            Mod_Input := Pi - Input;
         else
            Mod_Input := Input;
         end if;
```

```
      X_Squared := Mod_Input * Mod_Input;
      Inter_Result_1 := (((Cos_R_C9  * X_Squared +
                           Cos_R_C7) * X_Squared +
                           Cos_R_C5) * X_Squared +
                           Cos_R_C3) * X_Squared;
      Inter_Result_1 :=  Inter_Result_1 + 1.0 ;
      If Input > Pi_Over_2 then
         Inter_Result_1 := - Inter_Result_1;
      end if;
   else
      Inter_Result_0 := Real(Pi_Over_2 - Input);
      X_Squared := Inter_Result_0 * Inter_Result_0;
      Inter_Result_1 := (((Sin_R_C9  * X_Squared +
                           Sin_R_C7) * X_Squared +
                           Sin_R_C5) * X_Squared +
                           Sin_R_C3) * X_Squared;
      Inter_Result_1 := Inter_Result_1 * Inter_Result_0 + Inter_Result_0;
   end if;
   If Inter_Result_1 > 1.0 then
      Inter_Result_1 := 1.0;
   elsif Inter_Result_1 < -1.0 then
      Inter_Result_1 := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result_1 );
   return Result;
end Mod_Cos_R_5term;


function Mod_Cos_R_4term(Input : Radians) return Sin_Cos_Ratio is

   Inter_Result_0 : Real;
   Inter_Result_1 : Real;
   Mod_Input      : Radians;
   Result         : Sin_Cos_Ratio;
   X_Squared      : Real;

begin
   if (Input <= Pi_Over_4) or (Input >= 3.0 * Pi_Over_4) then
      if Input > Pi_Over_2 then
         Mod_Input := Pi - Input;
      else
         Mod_Input := Input;
      end if;
      X_Squared := Mod_Input * Mod_Input;
      Inter_Result_1 := ((Cos_R_C7  * X_Squared +
                          Cos_R_C5) * X_Squared +
                          Cos_R_C3) * X_Squared;
      Inter_Result_1 :=  Inter_Result_1 + 1.0 ;
      If Input > Pi_Over_2 then
         Inter_Result_1 := - Inter_Result_1;
      end if;
   else
      Inter_Result_0 := Real(Pi_Over_2 - Input);
      X_Squared := Inter_Result_0 * Inter_Result_0;
      Inter_Result_1 := ((Sin_R_C7  * X_Squared +
                          Sin_R_C5) * X_Squared +
                          Sin_R_C3) * X_Squared;
```

```
            Inter_Result_1 := Inter_Result_1 * Inter_Result_0 + Inter_Result_0;
         end if;
         If Inter_Result_1 > 1.0 then
            Inter_Result_1 := 1.0;
         elsif Inter_Result_1 < -1.0 then
            Inter_Result_1 := -1.0;
         end if;
         Result := Sin_Cos_Ratio( Inter_Result_1 );
         return Result;
      end Mod_Cos_R_4term;


-- -- Modified Taylor tangent functions

      function Mod_Tan_R_8term (Input : Radians) return Tan_Ratio is
      begin
         return Tan_Ratio(Sin_R_8term(Input)) /
                Tan_Ratio(Cos_R_8term(Input));
      end Mod_Tan_R_8term;


      function Mod_Tan_R_7term (Input : Radians) return Tan_Ratio is
      begin
         return Tan_Ratio(Sin_R_7term(Input)) /
                Tan_Ratio(Cos_R_7term(Input));
      end Mod_Tan_R_7term;


      function Mod_Tan_R_6term (Input : Radians) return Tan_Ratio is
      begin
         return Tan_Ratio(Sin_R_6term(Input)) /
                Tan_Ratio(Cos_R_6term(Input));
      end Mod_Tan_R_6term;


      function Mod_Tan_R_5term (Input : Radians) return Tan_Ratio is
      begin
         return Tan_Ratio(Sin_R_5term(Input)) /
                Tan_Ratio(Cos_R_5term(Input));
      end Mod_Tan_R_5term;


      function Mod_Tan_R_4term (Input : Radians) return Tan_Ratio is
      begin
         return Tan_Ratio(Sin_R_4term(Input)) /
                Tan_Ratio(Cos_R_4term(Input));
      end Mod_Tan_R_4term;

end Taylor_Radian_Operations;
```

3.3.6.8.9.7.9.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.7.9.1.8 LIMITATIONS

None.

3.3.6.8.9.7.9.1.9 LLCSC DESIGN

None.

3.3.6.8.9.7.9.1.10 UNIT DESIGN

None.

3.3.6.8.9.7.9.2 TAYLOR_DEGREE_OPERATIONS PACKAGE DESIGN (CATALOG #P841-0)

This generic package contains functions providing Taylor polynomial solutions for a set of trigonometric functions. This package is designed to handle units of degrees.

The following table lists the catalog numbers for subunits contained in this part:

Name	Catalog _#
Sin_D_8term	P842-0
Sin_D_7term	P843-0
Sin_D_6term	P844-0
Sin_D_5term	P845-0
Sin_D_4term	P867-0
Cos_D_8term	P846-0
Cos_D_7term	P847-0
Cos_D_6term	P848-0
Cos_D_5term	P849-0
Cos_D_4term	P850-0
Tan_D_8term	P851-0
Mod_Sin_D_8term	P852-0
Mod_Sin_D_7term	P853-0
Mod_Sin_D_6term	P854-0
Mod_Sin_D_5term	P855-0
Mod_Sin_D_4term	P856-0
Mod_Cos_D_8term	P857-0
Mod_Cos_D_7term	P858-0
Mod_Cos_D_6term	P859-0
Mod_Cos_D_5term	P860-0
Mod_Cos_D_4term	P861-0
Mod_Tan_D_8term	P862-0
Mod_Tan_D_7term	P863-0
Mod_Tan_D_6term	P864-0
Mod_Tan_D_5term	P865-0
Mod_Tan_D_4term	P866-0

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.7.9.2.1 REQUIREMENTS ALLOCATION

This part partially meets CAMP requirement R222.

3.3.6.8.9.7.9.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.7.9.2.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Degrees	Floating point	Allows floating point representation of degree measurements.
Real	Floating point	General floating point representation.
Sin_Cos_Ratio	Floating point	Represents sines and cosines.
Tan_Ratio	Floating point	Represents tangent values.

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
Pi	Degrees	constant	constant value Pi

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Overloaded operator to multiply degrees * degrees yielding a real result.

FORMAL PARAMETERS:

The following table describes the formal parameters for the functions contained in this part:

Function	Name	Type	Description
Sine Functions	Input	Degrees	Input for sine computation
Cosine Functions	Input	Degrees	Input for cosine computation
Tangent Functions	Input	Degrees	Input for tangent computation

3.3.6.8.9.7.9.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Sin_D_C3	constant	-31_348.4915	3rd term coefficient
Sin_D_C5	constant	145_551.339	5th term coefficient
Sin_D_C7	constant	-402_186_871.0	7th term coefficient
Sin_D_C9	constant	18_337_520_600.0	9th term coefficient
Sin_D_C11	constant	-547_254_221_000.0	11th term coefficient
Sin_D_C13	constant	11_508_293_600_000.0	13th term coefficient
Sin_D_C15	constant	-173_518_597_000_000.0	15th term coefficient
Cos_D_C2	constant	-1_641.40317	3rd term coefficient
Cos_D_C4	constant	449_134.064	5th term coefficient
Cos_D_C6	constant	-49_136_395.8	7th term coefficient
Cos_D_C8	constant	2_880_451_290.0	9th term coefficient
Cos_D_C10	constant	-105_066_264_000.0	11th term coefficient
Cos_D_C12	constant	2_612_971_200_000.0	13th term coefficient
Cos_D_C14	constant	-47_131_200_400_000.0	15th term coefficient
Tan_D_C3	constant	62_969.9829	3rd term coefficient
Tan_D_C5	constant	82_328_821.4	5th term coefficient
Tan_D_C7	constant	1.09349_829E11	7th term coefficient
Tan_D_C9	constant	1.45527_752E14	9th term coefficient
Tan_D_C11	constant	1.93616_001E17	11th term coefficient
Tan_D_C13	constant	2.57600_101E20	13th term coefficient
Tan_D_C15	constant	3.42729_477E23	15th term coefficient

3.3.6.8.9.7.9.2.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.7.9.2.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials.Taylor_Series)
package body Taylor_Degree_Operations is

-- -- The Sine constants are used in the Taylor operations for computing
   -- the sine.  The Cosine constants are used in computing cosines.  In
   -- the Modified Taylor operations, however, both sets of constants are
   -- used.  They are named to correspond to the power of X (Input) with
   -- which they are termed.

   Sin_D_C1  : constant :=  1.74532_92e-02;
   Sin_D_C3  : constant := -8.86096_158e-07;
   Sin_D_C5  : constant :=  1.34960_162e-11;
   Sin_D_C7  : constant := -9.78838_484e-17;
   Sin_D_C9  : constant :=  4.14126_699e-22;
   Sin_D_C11 : constant := -1.14680_931e-27;
   Sin_D_C13 : constant :=  2.23780_628e-33;
   Sin_D_C15 : constant := -3.13088_457e-39;

   Cos_D_C0  : constant :=  1.74532_92e-02;
   Cos_D_C2  : constant := -1.52308_710e-04;
   Cos_D_C4  : constant :=  3.86632_386e-09;
   Cos_D_C6  : constant := -3.92583_199e-14;
   Cos_D_C8  : constant :=  2.13549_430e-19;
   Cos_D_C10 : constant := -7.22787_516e-25;
   Cos_D_C12 : constant :=  1.66798_234e-30;
   Cos_D_C14 : constant := -2.79173_888e-36;

   Tan_D_C1  : constant := 1.74532_92e-02;
   Tan_D_C3  : constant := 1.77219_231e-06;
   Tan_D_C5  : constant := 2.15936_259e-10;
   Tan_D_C7  : constant := 2.66244_068e-14;
   Tan_D_C9  : constant := 3.28653_633e-18;
   Tan_D_C11 : constant := 4.05735_804e-22;
   Tan_D_C13 : constant := 5.00907_561e-26;
   Tan_D_C15 : constant := 6.18404_282e-30;


-- -- Taylor Sine functions

   function Sin_D_8term (Input : Degrees) return Sin_Cos_Ratio is

      Inter_result  : Real;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Real;

   begin
      X_Squared := Input * Input;
      Inter_Result :=  ((((((Sin_D_C15  * X_Squared +
                             Sin_D_C13) * X_Squared +
                             Sin_D_C11) * X_Squared +
                             Sin_D_C9)  * X_Squared +
                             Sin_D_C7)  * X_Squared +
                             Sin_D_C5)  * X_Squared +
```

```
                              Sin_D_C3)  * X_Squared;
      Inter_Result := Inter_Result * Real(Input) +
                (Degrees(Sin_D_C1) * Input);
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Sin_D_8term;


   function Sin_D_7term (Input : Degrees) return Sin_Cos_Ratio is

      Inter_result  : Real;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Real;

   begin
      X_Squared := Input * Input;
      Inter_Result :=  (((((Sin_D_C13  * X_Squared +
                            Sin_D_C11) * X_Squared +
                            Sin_D_C9)  * X_Squared +
                            Sin_D_C7)  * X_Squared +
                            Sin_D_C5)  * X_Squared +
                            Sin_D_C3)  * X_Squared;
      Inter_Result := Inter_Result * Real(Input) +
                (Degrees(Sin_D_C1) * Input);
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Sin_D_7term;


   function Sin_D_6term (Input : Degrees) return Sin_Cos_Ratio is

      Inter_result  : Real;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Real;

   begin
      X_Squared := Input * Input;
      Inter_Result :=  ((((Sin_D_C11 * X_Squared +
                           Sin_D_C9) * X_Squared +
                           Sin_D_C7) * X_Squared +
                           Sin_D_C5) * X_Squared +
                           Sin_D_C3) * X_Squared;
      Inter_Result := Inter_Result * Real(Input) +
                (Degrees(Sin_D_C1) * Input);
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
```

```
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Sin_D_6term;


   function Sin_D_5term (Input : Degrees) return Sin_Cos_Ratio is

      Inter_result  : Real;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Real;

   begin
      X_Squared := Input * Input;
      Inter_Result :=  (((Sin_D_C9  * X_Squared +
                         Sin_D_C7) * X_Squared +
                         Sin_D_C5) * X_Squared +
                         Sin_D_C3) * X_Squared;
      Inter_Result := Inter_Result * Real(Input) +
               (Degrees(Sin_D_C1) * Input);
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Sin_D_5term;


   function Sin_D_4term (Input : Degrees) return Sin_Cos_Ratio is

      Inter_result  : Real;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Real;

   begin
      X_Squared := Input * Input;
      Inter_Result :=  ((Sin_D_C7  * X_Squared +
                         Sin_D_C5) * X_Squared +
                         Sin_D_C3) * X_Squared;
      Inter_Result := Inter_Result * Real(Input) +
               (Degrees(Sin_D_C1) * Input);
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Sin_D_4term;


   -- -- Taylor Cosine functions

   function Cos_D_8term (Input : Degrees) return Sin_Cos_Ratio is
```

```
      Inter_result  : Real;
      Mod_Input     : Degrees;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Degrees;

   begin
      If Input > 90.0 then
         Mod_Input := 180.0 - Input;
      Else
         Mod_Input := Input;
      end if;
      X_Squared := Mod_Input * Mod_Input;
      Inter_Result := ((((((Cos_D_C14  * X_Squared +
                            Cos_D_C12) * X_Squared +
                            Cos_D_C10) * X_Squared +
                            Cos_D_C8) * X_Squared +
                            Cos_D_C6) * X_Squared +
                            Cos_D_C4) * X_Squared +
                            Cos_D_C2) * X_Squared;
      Inter_Result := Inter_Result + 1.0;
      If Input > 90.0 then
         Inter_Result := - Inter_Result;
      end if;
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Cos_D_8term;


   function Cos_D_7term (Input : Degrees) return Sin_Cos_Ratio is

      Inter_result  : Real;
      Mod_Input     : Degrees;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Degrees;

   begin
      If Input > 90.0 then
         Mod_Input := 180.0 - Input;
      Else
         Mod_Input := Input;
      end if;
      X_Squared := Mod_Input * Mod_Input;
      Inter_Result := (((((Cos_D_C12  * X_Squared +
                           Cos_D_C10) * X_Squared +
                           Cos_D_C8) * X_Squared +
                           Cos_D_C6) * X_Squared +
                           Cos_D_C4) * X_Squared +
                           Cos_D_C2) * X_Squared;
      Inter_Result := Inter_Result + 1.0;
      If Input > 90.0 then
         Inter_Result := - Inter_Result;
```

```
      end if;
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Cos_D_7term;


   function Cos_D_6term (Input : Degrees) return Sin_Cos_Ratio is

      Inter_result  : Real;
      Mod_Input     : Degrees;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Degrees;

   begin
      If Input > 90.0 then
         Mod_Input := 180.0 - Input;
      Else
         Mod_Input := Input;
      end if;
      X_Squared := Mod_Input * Mod_Input;
      Inter_Result := ((((Cos_D_C10  * X_Squared +
                          Cos_D_C8) * X_Squared +
                          Cos_D_C6) * X_Squared +
                          Cos_D_C4) * X_Squared +
                          Cos_D_C2) * X_Squared;
      Inter_Result := Inter_Result + 1.0;
      If Input > 90.0 then
         Inter_Result := - Inter_Result;
      end if;
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Cos_D_6term;


   function Cos_D_5term (Input : Degrees) return Sin_Cos_Ratio is

      Inter_result  : Real;
      Mod_Input     : Degrees;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Degrees;

   begin
      If Input > 90.0 then
         Mod_Input := 180.0 - Input;
      Else
         Mod_Input := Input;
      end if;
```

```
      X_Squared := Mod_Input * Mod_Input;
      Inter_Result := (((Cos_D_C8  * X_Squared +
                         Cos_D_C6) * X_Squared +
                         Cos_D_C4) * X_Squared +
                         Cos_D_C2) * X_Squared;
      Inter_Result := Inter_Result + 1.0;
      If Input > 90.0 then
         Inter_Result := - Inter_Result;
      end if;
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Cos_D_5term;


   function Cos_D_4term (Input : Degrees) return Sin_Cos_Ratio is

      Inter_result  : Real;
      Mod_Input     : Degrees;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Degrees;

   begin
      If Input > 90.0 then
         Mod_Input := 180.0 - Input;
      Else
         Mod_Input := Input;
      end if;
      X_Squared := Mod_Input * Mod_Input;
      Inter_Result := ((Cos_D_C6  * X_Squared +
                        Cos_D_C4) * X_Squared +
                        Cos_D_C2) * X_Squared;
      Inter_Result := Inter_Result + 1.0;
      If Input > 90.0 then
         Inter_Result := - Inter_Result;
      end if;
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Cos_D_4term;



-- -- Taylor Tangent fuctions

   function Tan_D_8term (Input : Degrees) return Tan_Ratio is

      Inter_result  : Real;
      Result        : Tan_Ratio;
```

```
      X_Squared     : Real;

   begin
      X_Squared := Input * Input;
      Inter_Result :=  ((((((Tan_D_C15  * X_Squared +
                             Tan_D_C13) * X_Squared +
                             Tan_D_C11) * X_Squared +
                             Tan_D_C9)  * X_Squared +
                             Tan_D_C7)  * X_Squared +
                             Tan_D_C5)  * X_Squared +
                             Tan_D_C3)  * X_Squared;
      Result := Tan_Ratio(Inter_Result) * Tan_Ratio(Input) +
                Tan_Ratio(Input) * Tan_D_C1;
      return Result;
   end Tan_D_8term;



-- -- Modified Taylor Sine functions

   function Mod_Sin_D_8term(Input : Degrees) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if abs(Input) >= 45.0 then
         Inter_Result_0 := Real(90.0 - Abs(Input));
         X_Squared := Inter_Result_0 * Inter_Result_0;
         Inter_Result_1 := ((((((Cos_D_C14  * X_Squared +
                                 Cos_D_C12) * X_Squared +
                                 Cos_D_C10) * X_Squared +
                                 Cos_D_C8)  * X_Squared +
                                 Cos_D_C6)  * X_Squared +
                                 Cos_D_C4)  * X_Squared +
                                 Cos_D_C2)  * X_Squared;
         Inter_Result_1 := Inter_Result_1 + 1.0;
         If Input <= - 45.0 then
            Inter_Result_1 := - Inter_Result_1;
         end if;
      else
         X_Squared := Input * Input;
         Inter_Result_1 := ((((((Sin_D_C15  * X_Squared +
                                 Sin_D_C13) * X_Squared +
                                 Sin_D_C11) * X_Squared +
                                 Sin_D_C9)  * X_Squared +
                                 Sin_D_C7)  * X_Squared +
                                 Sin_D_C5)  * X_Squared +
                                 Sin_D_C3)  * X_Squared;
         Inter_Result_1 := Inter_Result_1 * Real(Input) +
               (Degrees(Sin_D_C1) * Input);
      end if;
      if Inter_Result_1 > 1.0 then
         Inter_Result_1 := 1.0;
      elsif Inter_Result_1 < -1.0 then
```

```
        Inter_Result_1 := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result_1);
      return Result;
   end Mod_Sin_D_8term;


   function Mod_Sin_D_7term (Input : Degrees) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if abs(Input) >= 45.0 then
         Inter_Result_0 := Real(90.0 - Abs(Input));
         X_Squared := Inter_Result_0 * Inter_Result_0;
         Inter_Result_1 := (((((Cos_D_C12  * X_Squared +
                               Cos_D_C10) * X_Squared +
                               Cos_D_C8)  * X_Squared +
                               Cos_D_C6)  * X_Squared +
                               Cos_D_C4)  * X_Squared +
                               Cos_D_C2)  * X_Squared;
         Inter_Result_1 := Inter_Result_1 + 1.0;
         If Input <= - 45.0 then
            Inter_Result_1 := - Inter_Result_1;
         end if;
      else
         X_Squared := Input * Input;
         Inter_Result_1 := (((((Sin_D_C13  * X_Squared +
                               Sin_D_C11) * X_Squared +
                               Sin_D_C9)  * X_Squared +
                               Sin_D_C7)  * X_Squared +
                               Sin_D_C5)  * X_Squared +
                               Sin_D_C3)  * X_Squared;
         Inter_Result_1 := Inter_Result_1 * Real(Input) +
                 (Degrees(Sin_D_C1) * Input);
      end if;
      if Inter_Result_1 > 1.0 then
         Inter_Result_1 := 1.0;
      elsif Inter_Result_1 < -1.0 then
         Inter_Result_1 := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result_1 );
      return Result;
   end Mod_Sin_D_7term;


   function Mod_Sin_D_6term (Input : Degrees) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
```

```
      if abs(Input) >= 45.0 then
         Inter_Result_0 := Real(90.0 - Abs(Input));
         X_Squared := Inter_Result_0 * Inter_Result_0;
         Inter_Result_1 := (((((Cos_D_C10  * X_Squared +
                               Cos_D_C8) * X_Squared +
                               Cos_D_C6) * X_Squared +
                               Cos_D_C4) * X_Squared +
                               Cos_D_C2) * X_Squared;
         Inter_Result_1 := Inter_Result_1 + 1.0;
         If Input <= - 45.0 then
            Inter_Result_1 := - Inter_Result_1;
         end if;
      else
         X_Squared := Input * Input;
         Inter_Result_1 := ((((Sin_D_C11  * X_Squared +
                               Sin_D_C9) * X_Squared +
                               Sin_D_C7) * X_Squared +
                               Sin_D_C5) * X_Squared +
                               Sin_D_C3) * X_Squared;
         Inter_Result_1 := Inter_Result_1 * Real(Input) +
                (Degrees(Sin_D_C1) * Input);
      end if;
      if Inter_Result_1 > 1.0 then
         Inter_Result_1 := 1.0;
      elsif Inter_Result_1 < -1.0 then
         Inter_Result_1 := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result_1 );
      return Result;
   end Mod_Sin_D_6term;


   function Mod_Sin_D_5term (Input : Degrees) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if abs(Input) >= 45.0 then
         Inter_Result_0 := Real(90.0 - Abs(Input));
         X_Squared := Inter_Result_0 * Inter_Result_0;
         Inter_Result_1 := (((Cos_D_C8  * X_Squared +
                               Cos_D_C6) * X_Squared +
                               Cos_D_C4) * X_Squared +
                               Cos_D_C2) * X_Squared;
         Inter_Result_1 := Inter_Result_1 + 1.0;
         If Input <= - 45.0 then
            Inter_Result_1 := - Inter_Result_1;
         end if;
      else
         X_Squared := Input * Input;
         Inter_Result_1 := (((Sin_D_C9  * X_Squared +
                               Sin_D_C7) * X_Squared +
                               Sin_D_C5) * X_Squared +
                               Sin_D_C3) * X_Squared;
```

```
      Inter_Result_1 := Inter_Result_1 * Real(Input) +
            (Degrees(Sin_D_C1) * Input);
   end if;
   if Inter_Result_1 > 1.0 then
      Inter_Result_1 := 1.0;
   elsif Inter_Result_1 < -1.0 then
      Inter_Result_1 := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result_1 );
   return Result;
end Mod_Sin_D_5term;


function Mod_Sin_D_4term (Input : Degrees) return Sin_Cos_Ratio is

   Inter_Result_0 : Real;
   Inter_Result_1 : Real;
   Result         : Sin_Cos_Ratio;
   X_Squared      : Real;

begin
   if abs(Input) >= 45.0 then
      Inter_Result_0 := Real(90.0 - Abs(Input));
      X_Squared := Inter_Result_0 * Inter_Result_0;
      Inter_Result_1 := ((Cos_D_C6  * X_Squared +
                          Cos_D_C4) * X_Squared +
                          Cos_D_C2) * X_Squared;
      Inter_Result_1 := Inter_Result_1 + 1.0;
      If Input <= - 45.0 then
         Inter_Result_1 := - Inter_Result_1;
      end if;
   else
      X_Squared := Input * Input;
      Inter_Result_1 := ((Sin_D_C7  * X_Squared +
                          Sin_D_C5) * X_Squared +
                          Sin_D_C3) * X_Squared;
      Inter_Result_1 := Inter_Result_1 * Real(Input) +
            (Degrees(Sin_D_C1) * Input);
   end if;
   if Inter_Result_1 > 1.0 then
      Inter_Result_1 := 1.0;
   elsif Inter_Result_1 < -1.0 then
      Inter_Result_1 := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result_1 );
   return Result;
end Mod_Sin_D_4term;


-- -- Modified Taylor Cosine functions

   function Mod_Cos_D_8term(Input : Degrees) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Mod_Input      : Degrees;
      Result         : Sin_Cos_Ratio;
```

```
      X_Squared      : Real;

   begin
      if (Input <= 45.0) or (Input >= 135.0) then
         if Input > 90.0 then
            Mod_Input := 180.0 - Input;
         else
            Mod_Input := Input;
         end if;
         X_Squared := Mod_Input * Mod_Input;
         Inter_Result_1 := ((((((Cos_D_C14  * X_Squared +
                                 Cos_D_C12) * X_Squared +
                                 Cos_D_C10) * X_Squared +
                                 Cos_D_C8) * X_Squared +
                                 Cos_D_C6) * X_Squared +
                                 Cos_D_C4) * X_Squared +
                                 Cos_D_C2) * X_Squared;
         Inter_Result_1 :=  Inter_Result_1 + 1.0 ;
         If Input > 90.0 then
            Inter_Result_1 := - Inter_Result_1;
         end if;
      else
         Inter_Result_0 := Real(90.0 - Input);
         X_Squared := Inter_Result_0 * Inter_Result_0;
         Inter_Result_1 := ((((((Sin_D_C15  * X_Squared +
                                 Sin_D_C13) * X_Squared +
                                 Sin_D_C11) * X_Squared +
                                 Sin_D_C9) * X_Squared +
                                 Sin_D_C7) * X_Squared +
                                 Sin_D_C5) * X_Squared +
                                 Sin_D_C3) * X_Squared;
         Inter_Result_1 := Inter_Result_1 * Inter_Result_0 +
               (Sin_D_C1 * Inter_Result_0);
      end if;
      if Inter_Result_1 > 1.0 then
         Inter_Result_1 := 1.0;
      elsif Inter_Result_1 < -1.0 then
         Inter_Result_1 := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result_1 );
      return Result;
   end Mod_Cos_D_8term;


   function Mod_Cos_D_7term(Input : Degrees) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Mod_Input      : Degrees;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if (Input <= 45.0) or (Input >= 135.0) then
         if Input > 90.0 then
            Mod_Input := 180.0 - Input;
         else
```

```
            Mod_Input := Input;
         end if;
         X_Squared := Mod_Input * Mod_Input;
         Inter_Result_1 := (((((Cos_D_C12  * X_Squared +
                               Cos_D_C10) * X_Squared +
                               Cos_D_C8) * X_Squared +
                               Cos_D_C6) * X_Squared +
                               Cos_D_C4) * X_Squared +
                               Cos_D_C2) * X_Squared;
         Inter_Result_1 :=  Inter_Result_1 + 1.0 ;
         If Input > 90.0 then
            Inter_Result_1 := - Inter_Result_1;
         end if;
      else
         Inter_Result_0 := Real(90.0 - Input);
         X_Squared := Inter_Result_0 * Inter_Result_0;
         Inter_Result_1 := (((((Sin_D_C13  * X_Squared +
                               Sin_D_C11) * X_Squared +
                               Sin_D_C9) * X_Squared +
                               Sin_D_C7) * X_Squared +
                               Sin_D_C5) * X_Squared +
                               Sin_D_C3) * X_Squared;
         Inter_Result_1 := Inter_Result_1 * Inter_Result_0 +
                (Sin_D_C1 * Inter_Result_0);
      end if;
      if Inter_Result_1 > 1.0 then
         Inter_Result_1 := 1.0;
      elsif Inter_Result_1 < -1.0 then
         Inter_Result_1 := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result_1 );
      return Result;
   end Mod_Cos_D_7term;


   function Mod_Cos_D_6term(Input : Degrees) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Mod_Input      : Degrees;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if (Input <= 45.0) or (Input >= 135.0) then
         if Input > 90.0 then
            Mod_Input := 180.0 - Input;
         else
            Mod_Input := Input;
         end if;
         X_Squared := Mod_Input * Mod_Input;
         Inter_Result_1 := ((((Cos_D_C10  * X_Squared +
                               Cos_D_C8) * X_Squared +
                               Cos_D_C6) * X_Squared +
                               Cos_D_C4) * X_Squared +
                               Cos_D_C2) * X_Squared;
         Inter_Result_1 :=  Inter_Result_1 + 1.0 ;
```

```
      If Input > 90.0 then
         Inter_Result_1 := - Inter_Result_1;
      end if;
   else
      Inter_Result_0 := Real(90.0 - Input);
      X_Squared := Inter_Result_0 * Inter_Result_0;
      Inter_Result_1 := ((((Sin_D_C11  * X_Squared +
                            Sin_D_C9) * X_Squared +
                            Sin_D_C7) * X_Squared +
                            Sin_D_C5) * X_Squared +
                            Sin_D_C3) * X_Squared;
      Inter_Result_1 := Inter_Result_1 * Inter_Result_0 +
            (Sin_D_C1 * Inter_Result_0);
   end if;
   if Inter_Result_1 > 1.0 then
      Inter_Result_1 := 1.0;
   elsif Inter_Result_1 < -1.0 then
      Inter_Result_1 := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result_1 );
   return Result;
end Mod_Cos_D_6term;


function Mod_Cos_D_5term(Input : Degrees) return Sin_Cos_Ratio is

   Inter_Result_0 : Real;
   Inter_Result_1 : Real;
   Mod_Input      : Degrees;
   Result         : Sin_Cos_Ratio;
   X_Squared      : Real;

begin
   if (Input <= 45.0) or (Input >= 135.0) then
      if Input > 90.0 then
         Mod_Input := 180.0 - Input;
      else
         Mod_Input := Input;
      end if;
      X_Squared := Mod_Input * Mod_Input;
      Inter_Result_1 := (((Cos_D_C6  * X_Squared +
                           Cos_D_C6) * X_Squared +
                           Cos_D_C4) * X_Squared +
                           Cos_D_C2) * X_Squared;
      Inter_Result_1 :=  Inter_Result_1 + 1.0 ;
      If Input > 90.0 then
         Inter_Result_1 := - Inter_Result_1;
      end if;
   else
      Inter_Result_0 := Real(90.0 - Input);
      X_Squared := Inter_Result_0 * Inter_Result_0;
      Inter_Result_1 := (((Sin_D_C9  * X_Squared +
                           Sin_D_C7) * X_Squared +
                           Sin_D_C5) * X_Squared +
                           Sin_D_C3) * X_Squared;
      Inter_Result_1 := Inter_Result_1 * Inter_Result_0 +
            (Sin_D_C1 * Inter_Result_0);
```

```
      end if;
      if Inter_Result_1 > 1.0 then
         Inter_Result_1 := 1.0;
      elsif Inter_Result_1 < -1.0 then
         Inter_Result_1 := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result_1 );
      return Result;
   end Mod_Cos_D_5term;


   function Mod_Cos_D_4term(Input : Degrees) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Mod_Input      : Degrees;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if (Input <= 45.0) or (Input >= 135.0) then
         if Input > 90.0 then
            Mod_Input := 180.0 - Input;
         else
            Mod_Input := Input;
         end if;
         X_Squared := Mod_Input * Mod_Input;
         Inter_Result_1 := ((Cos_D_C6  * X_Squared +
                             Cos_D_C4) * X_Squared +
                             Cos_D_C2) * X_Squared;
         Inter_Result_1 :=  Inter_Result_1 + 1.0 ;
         If Input > 90.0 then
            Inter_Result_1 := - Inter_Result_1;
         end if;
      else
         Inter_Result_0 := Real(90.0 - Input);
         X_Squared := Inter_Result_0 * Inter_Result_0;
         Inter_Result_1 := ((Sin_D_C7  * X_Squared +
                             Sin_D_C5) * X_Squared +
                             Sin_D_C3) * X_Squared;
         Inter_Result_1 := Inter_Result_1 * Inter_Result_0 +
               (Sin_D_C1 * Inter_Result_0);
      end if;
      if Inter_Result_1 > 1.0 then
         Inter_Result_1 := 1.0;
      elsif Inter_Result_1 < -1.0 then
         Inter_Result_1 := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result_1 );
      return Result;
   end Mod_Cos_D_4term;


-- -- Modified Taylor Tangent functions

   function Mod_Tan_D_8term (Input : Degrees) return Tan_Ratio is
```

```
   begin
      return Tan_Ratio(Sin_D_8term(Input)) /
             Tan_Ratio(Cos_D_8term(Input));
   end Mod_Tan_D_8term;


   function Mod_Tan_D_7term (Input : Degrees) return Tan_Ratio is
   begin
      return Tan_Ratio(Sin_D_7term(Input)) /
             Tan_Ratio(Cos_D_7term(Input));
   end Mod_Tan_D_7term;


   function Mod_Tan_D_6term (Input : Degrees) return Tan_Ratio is
   begin
      return Tan_Ratio(Sin_D_6term(Input)) /
             Tan_Ratio(Cos_D_6term(Input));
   end Mod_Tan_D_6term;


   function Mod_Tan_D_5term (Input : Degrees) return Tan_Ratio is
   begin
      return Tan_Ratio(Sin_D_5term(Input)) /
             Tan_Ratio(Cos_D_5term(Input));
   end Mod_Tan_D_5term;


   function Mod_Tan_D_4term (Input : Degrees) return Tan_Ratio is
   begin
      return Tan_Ratio(Sin_D_4term(Input)) /
             Tan_Ratio(Cos_D_4term(Input));
   end Mod_Tan_D_4term;

end Taylor_Degree_Operations;
```

3.3.6.8.9.7.9.2.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.7.9.2.8 LIMITATIONS

None.

3.3.6.8.9.7.9.2.9 LLCSC DESIGN

None.

3.3.6.8.9.7.9.2.10 UNIT DESIGN

None.

3.3.6.8.9.7.9.3 TAYLOR_NATURAL_LOG PACKAGE DESIGN (CATALOG #P868-0)

This generic package contains functions providing Taylor polynomial solutions for the natural log function.

The following table lists the catalog numbers for subunits contained in this part:

Name	Catalog _#
Nat_Log_8term	P869-0
Nat_Log_7term	P870-0
Nat_Log_6term	P871-0
Nat_Log_5term	P872-0
Nat_Log_4term	P873-0

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.7.9.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R222.

3.3.6.8.9.7.9.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.7.9.3.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Inputs	Floating point	Floating point input to the function
Outputs	Floating point	Floating point output to the function

FORMAL PARAMETERS:

The following table describes the formal parameters for the functions contained in this part:

Function	Name	Type	Description
Natural Log Functions	Input	Floating Point	Input upon which to apply the funtion

3.3.6.8.9.7.9.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Nat_Log_C1	constant	2.0	Coefficient for 1st term
Nat_Log_C3	constant	0.66666_6666	Coefficient for 3rd term
Nat_Log_C5	constant	0.4	Coefficient for 5th term
Nat_Log_C7	constant	0.28574_1428	Coefficient for 7th term
Nat_Log_C9	constant	0.22222_2222	Coefficient for 9th term
Nat_Log_C11	constant	0.18181_8182	Coefficient for 11th term
Nat_Log_C13	constant	0.15384_6153	Coefficient for 13th term
Nat_Log_C15	constant	0.13333_3333	Coefficient for 15th term

3.3.6.8.9.7.9.3.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.7.9.3.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials.Taylor_Series)
package body Taylor_Natural_Log is

   Nat_Log_C1  : constant := 2.0;
   Nat_Log_C3  : constant := 0.66666_6666;
   Nat_Log_C5  : constant := 0.4;
   Nat_Log_C7  : constant := 0.28574_1428;
   Nat_Log_C9  : constant := 0.22222_2222;
   Nat_Log_C11 : constant := 0.18181_8182;
   Nat_Log_C13 : constant := 0.15384_6153;
   Nat_Log_C15 : constant := 0.13333_3333;


   function Nat_Log_8term ( Input : Inputs ) return Outputs is

      Inter_result  : Inputs;
      Result        : Outputs;
      Mod_Input     : Inputs;
      Mod_Squared   : Inputs;
```

```
   begin
      Mod_Input := (Input - 1.0)/(Input + 1.0);
      Mod_Squared := Mod_Input * Mod_Input;
      Inter_Result :=  (((((Nat_Log_C15  * Mod_Squared +
                            Nat_Log_C13) * Mod_Squared +
                            Nat_Log_C11) * Mod_Squared +
                            Nat_Log_C9)  * Mod_Squared +
                            Nat_Log_C7)  * Mod_Squared +
                            Nat_Log_C5)  * Mod_Squared +
                            Nat_Log_C3)  * Mod_Squared;
      Result := (Inter_Result * Mod_Input) + (Mod_Input * Nat_Log_C1);
      return Result;
   end Nat_Log_8term;


   function Nat_Log_7term ( Input : Inputs ) return Outputs is

      Inter_result  : Inputs;
      Result        : Outputs;
      Mod_Input     : Inputs;
      Mod_Squared   : Inputs;

   begin
      Mod_Input := (Input - 1.0)/(Input + 1.0);
      Mod_Squared := Mod_Input * Mod_Input;
      Inter_Result :=  ((((Nat_Log_C13  * Mod_Squared +
                           Nat_Log_C11) * Mod_Squared +
                           Nat_Log_C9)  * Mod_Squared +
                           Nat_Log_C7)  * Mod_Squared +
                           Nat_Log_C5)  * Mod_Squared +
                           Nat_Log_C3)  * Mod_Squared;
      Result := (Inter_Result * Mod_Input) + (Mod_Input * Nat_Log_C1);
      return Result;
   end Nat_Log_7term;


   function Nat_Log_6term ( Input : Inputs ) return Outputs is

      Inter_result  : Inputs;
      Result        : Outputs;
      Mod_Input     : Inputs;
      Mod_Squared   : Inputs;

   begin
      Mod_Input := (Input - 1.0)/(Input + 1.0);
      Mod_Squared := Mod_Input * Mod_Input;
      Inter_Result :=  (((Nat_Log_C11  * Mod_Squared +
                          Nat_Log_C9)  * Mod_Squared +
                          Nat_Log_C7)  * Mod_Squared +
                          Nat_Log_C5)  * Mod_Squared +
                          Nat_Log_C3)  * Mod_Squared;
      Result := (Inter_Result * Mod_Input) + (Mod_Input * Nat_Log_C1);
      return Result;
   end Nat_Log_6term;


   function Nat_Log_5term ( Input : Inputs ) return Outputs is
```

```
      Inter_result  : Inputs;
      Result        : Outputs;
      Mod_Input     : Inputs;
      Mod_Squared   : Inputs;

   begin
      Mod_Input := (Input - 1.0)/(Input + 1.0);
      Mod_Squared := Mod_Input * Mod_Input;
      Inter_Result :=  ((Nat_Log_C9  * Mod_Squared +
                         Nat_Log_C7) * Mod_Squared +
                         Nat_Log_C5) * Mod_Squared +
                         Nat_Log_C3) * Mod_Squared;
      Result := (Inter_Result * Mod_Input) + (Mod_Input * Nat_Log_C1);
      return Result;
   end Nat_Log_5term;


   function Nat_Log_4term ( Input : Inputs ) return Outputs is

      Inter_result  : Inputs;
      Result        : Outputs;
      Mod_Input     : Inputs;
      Mod_Squared   : Inputs;

   begin
      Mod_Input := (Input - 1.0)/(Input + 1.0);
      Mod_Squared := Mod_Input * Mod_Input;
      Inter_Result :=  ((Nat_Log_C7  * Mod_Squared +
                        Nat_Log_C5) * Mod_Squared +
                        Nat_Log_C3) * Mod_Squared;
      Result := (Inter_Result * Mod_Input) + (Mod_Input * Nat_Log_C1);
      return Result;
   end Nat_Log_4term;

end Taylor_Natural_Log;
```

3.3.6.8.9.7.9.3.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.7.9.3.8 LIMITATIONS

None.

3.3.6.8.9.7.9.3.9 LLCSC DESIGN

None.

3.3.6.8.9.7.9.3.10 UNIT DESIGN

None.

3.3.6.8.9.7.9.4 TAYLOR_LOG_BASE_N PACKAGE DESIGN (CATALOG #P874-0)

This packages contains generic functions providing Taylor polynomial solutions for the log function for base N.

The following table lists the catalog numbers for subunits contained in this part:

Name	Catalog _#
Log_Base_N_8term	P875-0
Log_Base_N_7term	P876-0
Log_Base_N_6term	P877-0
Log_Base_N_5term	P878-0
Log_Base_N_4term	P879-0

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.7.9.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R222.

3.3.6.8.9.7.9.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.7.9.4.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Inputs	Floating point	Floating point input to the function
Outputs	Floating point	Floating point output to the function

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
Base_N	Positive	defualt = 10	Base to operate in

FORMAL PARAMETERS:

The following table describes the formal parameters for the functions contained in this part:

Function	Name	Type	Description
Log Base N Functions	Input	Floating Point	Input upon which to apply the funtion

3.3.6.8.9.7.9.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Local_Natural_Log	Instantiated package	N/A	Natural log package used in calculating log base n

3.3.6.8.9.7.9.4.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.7.9.4.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials.Taylor_Series)
package body Taylor_Log_Base_N is

   package Local_Natural_Log is new Taylor_Natural_Log( Inputs  => Inputs,
                                                        Outputs => Outputs);

   package body Log_Base_N_8term is

      One_Over_Base_Log : constant Outputs := 1.0 /
                          Local_Natural_Log.Nat_Log_8term( Inputs(Base_N) );

      function Log_N_8term ( Input : Inputs ) return Outputs is
      begin
         return Local_Natural_Log.Nat_Log_8term( Input ) * One_Over_Base_Log;
      end Log_N_8term;
```

```
   end Log_Base_N_8term;

   package body Log_Base_N_7term is

      One_Over_Base_Log : constant Outputs := 1.0 /
                          Local_Natural_Log.Nat_Log_7term( Inputs(Base_N) );

      function Log_N_7term ( Input : Inputs ) return Outputs is
      begin
            return Local_Natural_Log.Nat_Log_7term( Input ) * One_Over_Base_Log;
      end Log_N_7term;

   end Log_Base_N_7term;

   package body Log_Base_N_6term is

      One_Over_Base_Log : constant Outputs := 1.0 /
                          Local_Natural_Log.Nat_Log_6term( Inputs(Base_N) );

      function Log_N_6term ( Input : Inputs ) return Outputs is
      begin
            return Local_Natural_Log.Nat_Log_6term( Input ) * One_Over_Base_Log;
      end Log_N_6term;

   end Log_Base_N_6term;

   package body Log_Base_N_5term is

      One_Over_Base_Log : constant Outputs := 1.0 /
                          Local_Natural_Log.Nat_Log_5term( Inputs(Base_N) );

      function Log_N_5term ( Input : Inputs ) return Outputs is
      begin
            return Local_Natural_Log.Nat_Log_5term( Input ) * One_Over_Base_Log;
      end Log_N_5term;

   end Log_Base_N_5term;

   package body Log_Base_N_4term is

      One_Over_Base_Log : constant Outputs := 1.0 /
                          Local_Natural_Log.Nat_Log_4term( Inputs(Base_N) );

      function Log_N_4term ( Input : Inputs ) return Outputs is
      begin
            return Local_Natural_Log.Nat_Log_4term( Input ) * One_Over_Base_Log;
      end Log_N_4term;

   end Log_Base_N_4term;

end Taylor_Log_Base_N;
```

3.3.6.8.9.7.9.4.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.7.9.4.8 LIMITATIONS

None.

3.3.6.8.9.7.9.4.9 LLCSC DESIGN

None.

3.3.6.8.9.7.9.4.10 UNIT DESIGN

None.

3.3.6.8.9.7.10 UNIT DESIGN

None.

(This page left intentionally blank.)

3.3.6.8.9.8 GENERAL_POLYNOMIAL PACKAGE DESIGN (CATALOG #P738-0)

This package allows the user to define a polynomial function and to then solve the user-polynomial for a given input value.

The following table lists the catalog numbers for subunits contained in this part:

Name	Catalog _#
Polynomial	P739-0

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.8.1 REQUIREMENTS ALLOCATION

This part partially meets CAMP requirement R214 through R222.

3.3.6.8.9.8.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.8.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Inputs	floating point type	Data type of independent values
Results	floating point type	Data type of dependent values

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Description
Coefficient_Count	Positive	Number of coefficient in the polynomial

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"**"	function	Exponential operator defining the operation: Inputs ** x := Results

FORMAL PARAMETERS:

The following table describes the formal parameters for the functions contained in this part:

Function	Name	Type	Description
Polynomial	Input	Inputs	Value of X for polynomial solution

3.3.6.8.9.8.4 LOCAL DATA

Data types:

The following chart describes the data types exported by this part:

Name	Range	Operators	Description
Coefficient_ Records	N/A	N/A	Contains the a and b components of a polynomial term: a*(x**b)
Table_ Dimensions	1 .. Coefficient_ Count	N/A N/A	Defines the size of the polynomial table

Data objects:

The following table describes the data objects exported by this part:

Name	Type	Definition
Polynomial_ Definition	array	Array of polynomial terms

3.3.6.8.9.8.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.8.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials)
package body General_Polynomial is

   function Polynomial (Input : Inputs) return Results is
      Result : Results;
   begin
      Result := 0.0;
      for Index in Table_Dimension
         loop
            Result := Result +
                      Polynomial_Definition( Index ).Coefficient *
                     (Input ** Polynomial_Definition( Index ).Power_of_X);
         end loop;
      return Result;
   end Polynomial;

end General_Polynomial;
```

3.3.6.8.9.8.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.8.8 LIMITATIONS

None.

3.3.6.8.9.8.9 LLCSC DESIGN

None.

3.3.6.8.9.8.10 UNIT DESIGN

None.

(This page left intentionally blank.)

3.3.6.8.9.9 SYSTEM FUNCTIONS PACKAGE DESIGN (CATALOG #P770-0)

This package provides access to the Ada system library for standard mathematical functions. For trigonometric functions, packages are provided to allow for inputs in units of radians, semicircles, or degrees.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.9.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R223.

3.3.6.8.9.9.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.3 INPUT/OUTPUT

None.

3.3.6.8.9.9.4 LOCAL DATA

None.

3.3.6.8.9.9.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials)
package body System_Functions is

   package body Radian_Operations is separate;

   package body Semicircle_Operations is separate;

   package body Degree_Operations is separate;

   package body Square_Root is separate;

   package body Base_10_Logarithm is separate;

   package body Base_N_Logarithm is separate;

end System_Functions;
```

3.3.6.8.9.9.7 UTILIZATION OF OTHER ELEMENTS

The following library units are with'd by this part:
1. Math_Lib

3.3.6.8.9.9.8 LIMITATIONS

None.

3.3.6.8.9.9.9 LLCSC DESIGN

3.3.6.8.9.9.9.1 RADIAN OPERATIONS PACKAGE DESIGN (CATALOG #P771-0)

This package contains a set of trigonometric functions which deal with angles in units of radians. The functions provided are sine, cosine, tangent, arcsine, arccosine, arctangent.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.9.9.1.1 REQUIREMENTS ALLOCATION

Together with the other parts in the System_Functions package, this part meets CAMP requirement R223.

3.3.6.8.9.9.9.1.2 LOCAL ENTITIES DESIGN

Packages:

The following describes the packages contained in this part:

Name	Type	Description
Radian_Math_Lib	package	Math library where functions called will have inputs and outputs of type Radians; input and output values will be converted as necessary

3.3.6.8.9.9.9.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined in the package specification of Radian_Operations:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Radians	floating point type	Data type describing units of angles
Sin_Cos_Ratio	floating point type	Data type describing output values from sine and cosine functions and input values to arcsine and arccosine functions
Tan_Ratio	floating point type	Data type describing output values from tangent function and input values to arctangent function

3.3.6.8.9.9.9.1.4 LOCAL DATA

None.

3.3.6.8.9.9.9.1.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.1.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials.System_Functions)
package body Radian_Operations is

-- -------------------------
-- --instantiated packages-
-- -------------------------

   package Radian_Math_Lib is new Math_Lib (Real => Radians);

   package M_Lib renames Radian_Math_Lib;

-- ----------------------------------------
-- --renamed functions within Local_Math_Lib
-- ----------------------------------------

   function Ada_Sin (Input : Radians) return Radians renames M_Lib.Sin;
   function Ada_Cos (Input : Radians) return Radians renames M_Lib.Cos;
   function Ada_Tan (Input : Radians) return Radians renames M_Lib.Tan;
   function Ada_Arcsin
               (Input : Radians) return Radians renames M_Lib.Asin;
   function Ada_Arccos
               (Input : Radians) return Radians renames M_Lib.Acos;
   function Ada_Arctan
               (Input : Radians) return Radians renames M_Lib.Atan;

end Radian_Operations;
```

3.3.6.8.9.9.9.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.9.9.1.8 LIMITATIONS

None.

3.3.6.8.9.9.9.1.9 LLCSC DESIGN

None.

3.3.6.8.9.9.9.1.10 UNIT DESIGN

3.3.6.8.9.9.9.1.10.1 SIN UNIT DESIGN (CATALOG #P772-0)

This function returns the sine of an angle with units of radians.

3.3.6.8.9.9.9.1.10.1.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.8.9.9.9.1.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.9.1.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Radians	In	Angle for which a sine is desired

3.3.6.8.9.9.9.1.10.1.4 LOCAL DATA

None.

3.3.6.8.9.9.9.1.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.1.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
function Sin (Input : Radians) return Sin_Cos_Ratio is

begin

   return Sin_Cos_Ratio(Ada_Sin(Input));

exception

   when M_Lib.ROprand => raise Invalid_Operand;

end Sin;
```

3.3.6.8.9.9.9.1.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Radian_Operations:

Name	Type	Description
Radians	floating point type	Data type describing units of angles
Sin_Cos_Ratio	floating point type	Data type describing output values from sine and cosine functions and input values to arcsine and arccosine functions

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of System_Functions:

Name	Description
Invalid_Operand	The input value is in an improper format not accepted by the operating system

Subprograms and task entries:

The following table describes the subprograms required by this part and defined in the package body of Radian_Operations:

Name	Type	Description
Ada_Sin	function	Sine function handling units of radians

3.3.6.8.9.9.9.1.10.1.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Operand	Raised if the format of the input is invalid and not accepted by the operating system.

3.3.6.8.9.9.9.1.10.2 COS UNIT DESIGN (CATALOG #P773-0)

This function returns the cosine of an angle with units of radians.

3.3.6.8.9.9.9.1.10.2.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.8.9.9.9.1.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.9.1.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Radians	In	Angle for which a cosine is desired

3.3.6.8.9.9.9.1.10.2.4 LOCAL DATA

None.

3.3.6.8.9.9.9.1.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.1.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
function Cos (Input : Radians) return Sin_Cos_Ratio is

begin

   return Sin_Cos_Ratio(Ada_Cos(Input));

exception

   when M_Lib.ROprand => raise Invalid_Operand;

end Cos;
```

3.3.6.8.9.9.9.1.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Radian_Operations:

Name	Type	Description
Radians	floating point type	Data type describing units of angles
Sin_Cos_Ratio	floating point type	Data type describing output values from sine and cosine functions and input values to arcsine and arccosine functions

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of System_Functions:

Name	Description
Invalid_Operand	The input value is in an improper format not accepted by the operating system

Subprograms and task entries:

The following table describes the subprograms required by this part and defined in the package body of Radian_Operations:

Name	Type	Description
Ada_Cos	function	Cosine function handling units of radians

3.3.6.8.9.9.9.1.10.2.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Operand	Raised if the format of the input value is invalid and not accepted by the operating system

3.3.6.8.9.9.9.1.10.3 TAN UNIT DESIGN (CATALOG #P774-0)

This function returns the tangent of an angle with units of Radians.

3.3.6.8.9.9.9.1.10.3.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.8.9.9.9.1.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.9.1.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Radians	In	Angle for which a tangent is desired

3.3.6.8.9.9.9.1.10.3.4 LOCAL DATA

None.

3.3.6.8.9.9.9.1.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.1.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
function Tan (Input : Radians) return Tan_Ratio is

begin

   return Tan_Ratio(Ada_Tan(Input));

exception

   when M_Lib.ROprand   => raise Invalid_Operand;
   when M_Lib.FloOveMat => raise Overflow;

end Tan;
```

3.3.6.8.9.9.9.1.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Radian_Operations:

Name	Type	Description
Radians	floating point type	Data type describing units of angles
Tan_Ratio	floating point type	Data type describing output values from tangent function and input values to arctangent function

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of System_Functions:

Name	Description
Invalid_Operand	The input value is in an improper format not accepted by the operating system
Overflow	A floating point overflow was encountered during the calculations

Subprograms and task entries:

The following table describes the subprograms required by this part and defined in the package body of Radian_Operations:

Name	Type	Description
Ada_Tan	function	Tangent function handling units of radians

3.3.6.8.9.9.9.1.10.3.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Operand	Raised if the input value has an improper format which is not accepted by the operating system
Overflow	Raised if a floating point overflow error is encountered during computations

3.3.6.8.9.9.9.1.10.4 ARCSIN UNIT DESIGN (CATALOG #P775-0)

This function returns the Arcsin, in units of Radians, of an input value. The input value must not be greater than 1.0 or less than -1.0.

3.3.6.8.9.9.9.1.10.4.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.8.9.9.9.1.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.9.1.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Sin_Cos_Ratio	In	Value which an arcsine is desired

3.3.6.8.9.9.9.1.10.4.4 LOCAL DATA

None.

3.3.6.8.9.9.9.1.10.4.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.1.10.4.6 PROCESSING

The following describes the processing performed by this part:

```
function Arcsin (Input : Sin_Cos_Ratio) return Radians is

begin

   return Ada_Arcsin(Radians(Input));

exception

   when M_Lib.ROprand   => raise Invalid_Operand;
   when M_Lib.InvArgMat => raise Invalid_Argument;

end Arcsin;
```

3.3.6.8.9.9.9.1.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Radian_Operations:

Name	Type	Description
Radians	floating point type	Data type describing units of angles
Sin_Cos_Ratio	floating point type	Data type describing output values from sine and cosine functions and input values to arcsine and arccosine functions

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of System_Functions:

Name	Description
Invalid_Operand	The input value is in an improper format not accepted by the operating system
Invalid_Argument	The input value is an a range unacceptable to the function being called

Subprograms and task entries:

The following table describes the subprograms required by this part and defined in the package body of Radian_Operations:

Name	Type	Description
Ada_Arcsin	function	Arcsine function handling units of radians

3.3.6.8.9.9.9.1.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Operand	Raised if the input value is in an improper format that is not accepted by the operating system
Invalid_Argument	Raised if the absolute value of the input is greater then 1.0

3.3.6.8.9.9.9.1.10.5 ARCCOS UNIT DESIGN (CATALOG #P776-0)

This function returns the Arccos, in units of Radians, of an input value. The absolute value of the input must not be greater than 1.0.

3.3.6.8.9.9.9.1.10.5.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.8.9.9.9.1.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.9.1.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Sin_Cos_Ratio	In	Value which an arccosine is desired

3.3.6.8.9.9.9.1.10.5.4 LOCAL DATA

None.

3.3.6.8.9.9.9.1.10.5.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.1.10.5.6 PROCESSING

The following describes the processing performed by this part:

```
function Arccos (Input : Sin_Cos_Ratio) return Radians is

begin

   return Ada_Arccos(Radians(input));

exception

   when M_Lib.ROprand   => raise Invalid_Operand;
   when M_Lib.InvArgMat => raise Invalid_Argument;

end Arccos;
```

3.3.6.8.9.9.9.1.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Radian_Operations:

Name	Type	Description
Radians	floating point type	Data type describing units of angles
Sin_Cos_Ratio	floating point type	Data type describing output values from sine and cosine functions and input values to arcsine and arccosine functions

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of System_Functions:

Name	Description
Invalid_Operand	The input value is in an improper format not accepted by the operating system
Invalid_Argument	The input value is an a range unacceptable to the function being called

Subprograms and task entries:

The following table describes the subprograms required by this part and defined in the package body of Radian_Operations:

Name	Type	Description
Ada_Arccos	function	Arccosine function handling units of radians

3.3.6.8.9.9.9.1.10.5.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Operand	Raised if the input value is in an improper format that is not accepted by the operating system
Invalid_Argument	Raised if the absolute value of the input is greater than 1.0

3.3.6.8.9.9.9.1.10.6 ARCTAN UNIT DESIGN (CATALOG #P777-0)

This function returns the Arctangent, in units of Radians, of an input value.

3.3.6.8.9.9.9.1.10.6.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.8.9.9.9.1.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.9.1.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Tan_Ratio	In	Value which an arctangent is desired

3.3.6.8.9.9.9.1.10.6.4 LOCAL DATA

None.

3.3.6.8.9.9.9.1.10.6.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.1.10.6.6 PROCESSING

The following describes the processing performed by this part:

```
function Arctan (Input : Tan_Ratio) return Radians is
begin
   return Ada_Arctan(Radians(input));
exception
   when M_Lib.ROprand => raise Invalid_Operand;
end Arctan;
```

3.3.6.8.9.9.9.1.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Radian_Operations:

Name	Type	Description
Radians	floating point type	Data type describing units of angles
Tan_Ratio	floating point type	Data type describing output values from tangent function and input values to arctangent function

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of System_Functions:

Name	Description
Invalid_Operand	The input value is in an improper format not accepted by the operating system

Subprograms and task entries:

The following table describes the subprograms required by this part and defined in the package body of Radian_Operations:

Name	Type	Description
Ada_Arctan	function	Arctangent function handling units of radians

3.3.6.8.9.9.9.1.10.6.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Operand	Raised if the input value is in an improper format which is not accepted by the operating system

3.3.6.8.9.9.9.2 SEMICIRCLE OPERATIONS PACKAGE DESIGN (CATALOG #P778-0)

This package contains a set of trigonometric functions which deal with angles in units of semicircles. The functions provided are sine, cosine, tangent, arcsine, arccosine, arctangent.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.9.9.2.1 REQUIREMENTS ALLOCATION

Together with the other parts in the System_Functions package, this part meets CAMP requirement R223.

3.3.6.8.9.9.9.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.9.2.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously described at the package specification level of Semicircle_Operations:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Scalars	floating point type	Describes data type of input object pi
Semicircles	floating point type	Data type describing units of angles
Sin_Cos_Ratio	floating point type	Data type describing output values from sine and cosine functions and input values to arcsine and arccosine functions
Tan_Ratio	floating point type	Data type describing output values from tangent function and input values to arctangent function

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
Pi	Scalars	N/A	Number of radians in a semicircle

Subprograms:

The following table describes the generic formal subroutines (operators) required by this part:

Name	Left Input Type	Right Input Type	Result Type
"*"	Semicircles	Scalars	Scalars
"*"	Scalars	Scalars	Semicircles
"/"	Scalars	Scalars	Scalars

3.3.6.8.9.9.9.2 4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
One_Over_Pi	Scalars	Contains the value 1/pi

3.3.6.8.9.9.9.2.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.2.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials.System_Functions)
package body Semicircle_Operations is

-- ------------------------
-- --instantiated packages-
-- ------------------------

   package Semicircle_Math_Lib is new Math_Lib (Real => Scalars);

   package M_Lib renames Semicircle_Math_Lib;

-- ----------------------------------------
-- --renamed functions within Local_Math_Lib
-- ----------------------------------------

   function Ada_Sin    (Input : Scalars) return Scalars renames M_Lib.Sin;
   function Ada_Cos    (Input : Scalars) return Scalars renames M_Lib.Cos;
   function Ada_Tan    (Input : Scalars) return Scalars renames M_Lib.Tan;
   function Ada_Arcsin
               (Input : Scalars) return Scalars renames M_Lib.Asin;
   function Ada_Arccos
               (Input : Scalars) return Scalars renames M_Lib.Acos;
   function Ada_Arctan
               (Input : Scalars) return Scalars renames M_Lib.Atan;
```

```
-- --------------------
-- --local declarations
-- --------------------

   One_Over_Pi : constant Scalars := 1.0 / Pi;

end Semicircle_Operations;
```

3.3.6.8.9.9.9.2.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.9.9.2.8 LIMITATIONS

None.

3.3.6.8.9.9.9.2.9 LLCSC DESIGN

None.

3.3.6.8.9.9.9.2.10 UNIT DESIGN

3.3.6.8.9.9.9.2.10.1 SIN UNIT DESIGN (CATALOG #P779-0)

This function returns the sine of an angle with units of semicircles.

3.3.6.8.9.9.9.2.10.1.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.8.9.9.9.2.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.9.2.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Semicircles	In	Angle for which a sine is desired

3.3.6.8.9.9.9.2.10.1.4 LOCAL DATA

None.

3.3.6.8.9.9.9.2.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.2.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
function Sin (Input : Semicircles) return Sin_Cos_Ratio is
begin
   return Sin_Cos_Ratio(Ada_Sin(input*pi));
exception
   when M_Lib.ROprand => raise Invalid_Operand;
end Sin;
```

3.3.6.8.9.9.9.2.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Semicircle_Operations:

Name	Type	Description
Semicircles	floating point type	Data type describing units of angles
Sin_Cos_Ratio	floating point type	Data type describing output values from sine and cosine functions and input values to arcsine and arccosine functions

Data objects:

The following table summarizes the generic objects required by this part and defined at the package specification level of Semicircle_Operations:

Name	Type	Value	Description
Pi	Scalars	N/A	Number of radians in a semicircle

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of System_Functions:

Name	Description
Invalid_Operand	The input value is in an improper format not accepted by the operating system

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Semicircle_Operations:

Name	Type	Description
Ada_Sin	function	Sine function handling units of radians

3.3.6.8.9.9.9.2.10.1.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Operand	Raised if the format of the input is such that after conversion to an angle with units of radians is invalid and not accepted by the operating system

3.3.6.8.9.9.9.2.10.2 COS UNIT DESIGN (CATALOG #P1087-0)

This function returns the cosine of an angle with units of semicircles.

3.3.6.8.9.9.9.2.10.2.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.8.9.9.9.2.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.9.2.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Semicircles	In	Angle for which a cosine is desired

3.3.6.8.9.9.9.2.10.2.4 LOCAL DATA

None.

3.3.6.8.9.9.9.2.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.2.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
function Cos (Input : Semicircles) return Sin_Cos_Ratio is
begin
   return Sin_Cos_Ratio(Ada_Cos(input*pi));
exception
   when M_Lib.ROPrand => raise Invalid_Operand;
end Cos;
```

3.3.6.8.9.9.9.2.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Semicircle_Operations:

Name	Type	Description
Semicircles	floating point type	Data type describing units of angles
Sin_Cos_Ratio	floating point type	Data type describing output values from sine and cosine functions and input values to arcsine and arccosine functions

Data objects:

The following table summarizes the generic objects required by this part and defined at the package specification level of Semicircle_Operations:

Name	Type	Value	Description
Pi	Scalars	N/A	Number of radians in a semicircle

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of System_Functions:

Name	Description
Invalid_Operand	The input value is in an improper format not accepted by the operating system

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Semicircle_Operations:

Name	Type	Description
Ada_Cos	function	Cosine function handling units of radians

3.3.6.8.9.9.9.2.10.2.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Operand	Raised if the format of the input is such that after conversion to an angle with units of radians is invalid and not accepted by the operating system

3.3.6.8.9.9.9.2.10.3 TAN UNIT DESIGN (CATALOG #P781-0)

This function returns the tangent of an angle with units of Semicircles.

3.3.6.8.9.9.9.2.10.3.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.8.9.9.9.2.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.9.2.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Semicircles	In	Angle for which a tangent is desired

3.3.6.8.9.9.9.2.10.3.4 LOCAL DATA

None.

3.3.6.8.9.9.9.2.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.2.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
function Tan (Input : Semicircles) return Tan_Ratio is

begin

   return Tan_Ratio(Ada_Tan(input*pi));

exception

   when M_Lib.ROprand   => raise Invalid_Operand;
   when M_Lib.FloOveMat => raise Overflow;

end Tan;
```

3.3.6.8.9.9.9.2.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

Semicircle_Operations:

Name	Type	Description
Semicircles	floating point type	Data type describing units of angles
Tan_Ratio	floating point type	Data type describing output values from tangent function and input values to arctangent function

Data objects:

The following table summarizes the generic objects required by this part and defined at the package specification level of Semicircle_Operations:

Name	Type	Value	Description
Pi	Scalars	N/A	Number of radians in a semicircle

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of System_Functions:

Name	Description
Invalid_Operand	The input value is in an improper format not accepted by the operating system
Overflow	A floating point overflow was encountered during the calculations

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Semicircle_Operations:

Name	Type	Description
Ada_Tan	function	Tangent function handling units of radians

3.3.6.8.9.9.9.2.10.3.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Operand	Raised if the format of the input is such that after unit and data type conversion it is invalid and not accepted by the operating system
Overflow	Raised if a floating point overflow error is encountered during computations

3.3.6.8.9.9.9.2.10.4 ARCSIN UNIT DESIGN (CATALOG #P782-0)

This function calculates the arcsine of an input value with the result being in units of semicircles. The absolute value of the input must not be greater than 1.0.

3.3.6.8.9.9.9.2.10.4.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.8.9.9.9.2.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.9.2.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
input	Sin_Cos_Ratio	In	Value for which an arcsine is desired

3.3.6.8.9.9.9.2.10.4.4 LOCAL DATA

None.

3.3.6.8.9.9.9.2.10.4.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.2.10.4.6 PROCESSING

The following describes the processing performed by this part:

```
function Arcsin (Input : Sin_Cos_Ratio) return Semicircles is

begin

   return Ada_Arcsin(Scalars(input)) * One_Over_Pi;

exception

   when M_Lib.ROprand   => raise Invalid_Operand;
   when M_Lib.InvArgMat => raise Invalid_Argument;

end Arcsin;
```

3.3.6.8.9.9.9.2.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Semicircle_Operations:

Name	Type	Description
Scalars	floating	Describes data type of input object pi
Semicircles	floating point type	Data type describing units of angles
Sin_Cos_Ratio	floating point type	Data type describing output values from sine and cosine functions and input values to arcsine and arccosine functions

Data objects:

The following table summarizes the generic objects required by this part and defined at the package specification level of Semicircle_Operations:

Name	Type	Value	Description
Pi	Scalars	N/A	Number of radians in a semicircle

The following table summarizes the objects required by this part and defined in the package body of Semicircle_Operations:

Name	Type	Value	Description
One_Over_Pi	Scalars	1/pi	Number of radians in a semicircle

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of System_Functions:

Name	Description
Invalid_Operand	The input value is in an improper format not accepted by the operating system
Invalid_Argument	The input value is an a range unacceptable to the function being called

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Semicircle_Operations:

Name	Type	Description
Ada_Arcsin	function	Arcsine function handling units of radians

3.3.6.8.9.9.9.2.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Operand	Raised if the format of the input value is such that after data type conversion it is invalid and not accepted by the operating system
Invalid_Argument	Raised if the absolute value of the input value is greater than 1.0

3.3.6.8.9.9.9.2.10.5 ARCCOS UNIT DESIGN (CATALOG #P783-0)

This function returns the Arccos, in units of Semicircles, of an input value. The absolute value of the input must not be greater than 1.0.

3.3.6.8.9.9.9.2.10.5.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.8.9.9.9.2.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.9.2.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Sin_Cos_Ratio	In	Value for which an arccosine is desired

3.3.6.8.9.9.9.2.10.5.4 LOCAL DATA

None.

3.3.6.8.9.9.9.2.10.5.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.2.10.5.6 PROCESSING

The following describes the processing performed by this part:

```
function Arccos (Input : Sin_Cos_Ratio) return Semicircles is

begin

   return Ada_Arccos(Scalars(Input)) * One_Over_Pi;

exception

   when M_Lib.ROprand   => raise Invalid_Operand;
   when M_Lib.InvArgMat => raise Invalid_Argument;

end Arccos;
```

3.3.6.8.9.9.9.2.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Semicircle_Operations:

Name	Type	Description
Scalars	floating	Describes data type of input object pi
Semicircles	floating point type	Data type describing units of angles
Sin_Cos_Ratio	floating point type	Data type describing output values from sine and cosine functions and input values to arcsine and arccosine functions

Data objects:

The following table summarizes the generic objects required by this part and defined at the package specification level of Semicircle_Operations:

Name	Type	Value	Description
Pi	Scalars	N/A	Number of radians in a semicircle

The following table summarizes the objects required by this part and defined in the package body of Semicircle_Operations:

Name	Type	Value	Description
One_Over_Pi	Scalars	1/pi	Number of radians in a semicircle

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of System_Functions:

Name	Description
Invalid_Operand	The input value is in an improper format not accepted by the operating system
Invalid_Argument	The input value is an a range unacceptable to the function being called

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Semicircle_Operations:

Name	Type	Description
Ada_Arccos	function	Arccosine function handling units of radians

3.3.6.8.9.9.9.2.10.5.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Operand	Raised if the format of the input value is such that it is invalid or not accepted by the operating system after it has been converted to a data type of Scalars
Invalid_Argument	Raised if the absolute value of the input is greater than 1.0

3.3.6.8.9.9.9.2.10.6 ARCTAN UNIT DESIGN (CATALOG #P784-0)

This function returns the arctangent, in units of semicircles, of an input value.

3.3.6.8.9.9.9.2.10.6.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.8.9.9.9.2.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.9.2.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Tan_Ratio	In	Value for which an arctangent is desired

3.3.6.8.9.9.9.2.10.6.4 LOCAL DATA

None.

3.3.6.8.9.9.9.2.10.6.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.2.10.6.6 PROCESSING

The following describes the processing performed by this part:

```
function Arctan (Input : Tan_Ratio) return Semicircles is

begin

   return Ada_Arctan(Scalars(Input)) * One_Over_Pi;

exception

   when M_Lib.ROprand => raise Invalid_Operand;

end Arctan;
```

3.3.6.8.9.9.9.2.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Semicircle_Operations:

Name	Type	Description
Scalars	floating	Describes data type of input object pi
Semicircles	floating point type	Data type describing units of angles
Tan_Ratio	floating point type	Data type describing output values from tangent function and input values to arctangent function

Data objects:

The following table summarizes the generic objects required by this part and defined at the package specification level of Semicircle_Operations:

Name	Type	Value	Description
Pi	Scalars	N/A	Number of radians in a semicircle

The following table summarizes the objects required by this part and defined in the package body of Semicircle_Operations:

Name	Type	Value	Description
One_Over_Pi	Scalars	1/pi	Number of radians in a semicircle

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of System_Functions:

Name	Description
Invalid_Operand	The input value is in an improper format not accepted by the operating system

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Semicircle_Operations:

Name	Type	Description
Ada_Arctan	function	Arctangent function handling units of radians

3.3.6.8.9.9.9.2.10.6.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Operand	Raised if the input value is in an improper format which is not accepted by the operating system

3.3.6.8.9.9.9.3 DEGREE_OPERATIONS PACKAGE DESIGN (CATALOG #P785-0)

This package contains a set of trigonometric functions which deal with angles in units of degrees. The functions provided are sine, cosine, tangent, arcsine, arccosine, and arctangent.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.9.9.3.1 REQUIREMENTS ALLOCATION

Together with other parts in the System_Functions package, this part meets CAMP requirement R223.

3.3.6.8.9.9.9.3.2 LOCAL ENTITIES DESIGN

Packages:

The following table describes the packages local to this part:

Name	Type	Description
Degree_Math_Lib	package	Math library where functions called will have inputs and outputs of type Degrees; input and output values will be converted as necessary

3.3.6.8.9.9.9.3.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined in this part's package specification:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Degrees	floating point type	Data type describing units of angles
Sin_Cos_Ratio	floating point type	Data type describing output values from sine and cosine functions and input values to arcsine and arccosine functions
Tan_Ratio	floating point type	Data type describing output values from tangent function and input values to arctangent function

3.3.6.8.9.9.9.3.4 LOCAL DATA

None.

3.3.6.8.9.9.9.3.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.3.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials.System_Functions)
package body Degree_Operations is

-- ----------------------
-- --instantiated package-
-- ----------------------

   package Degree_Math_Lib is new Math_Lib (Real => Degrees);

   package M_Lib renames Degree_Math_Lib;

-- ----------------------------------------
-- --renamed functions within Degree_Math_Lib
-- ----------------------------------------

   function Ada_Sin    (Input : Degrees)
                       return Degrees renames M_Lib.SinD;
   function Ada_Cos    (Input : Degrees)
                       return Degrees renames M_Lib.CosD;
   function Ada_Tan    (Input : Degrees)
                       return Degrees renames M_Lib.TanD;
   function Ada_Arcsin (Input : Degrees)
                       return Degrees renames M_Lib.AsinD;
   function Ada_Arccos (Input : Degrees)
                       return Degrees renames M_Lib.AcosD;
   function Ada_Arctan (Input : Degrees)
                       return Degrees renames M_Lib.AtanD;

end Degree_Operations;
```

3.3.6.8.9.9.9.3.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.9.9.3.8 LIMITATIONS

None.

3.3.6.8.9.9.9.3.9 LLCSC DESIGN

None.

3.3.6.8.9.9.9.3.10 UNIT DESIGN

3.3.6.8.9.9.9.3.10.1 SIN UNIT DESIGN (CATALOG #P786-0)

This functions returns the sine of an angle with units of degrees.

3.3.6.8.9.9.9.3.10.1.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.8.9.9.9.3.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.9.3.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Degrees	In	Angle for which a sine is desired

3.3.6.8.9.9.9.3.10.1.4 LOCAL DATA

None.

3.3.6.8.9.9.9.3.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.3.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
function Sin (Input : Degrees) return Sin_Cos_Ratio is

begin

   return Sin_Cos_Ratio(Ada_Sin(input));

exception

   when M_Lib.ROprand    => raise Invalid_Operand;
   when M_Lib.FloUndMat => raise Underflow;

end Sin;
```

3.3.6.8.9.9.9.3.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the types required by this part and defined as generic parameters for the package specification of Degree_Operations:

Name	Type	Description
Degrees	floating point type	Data type describing units of angles
Sin_Cos_Ratio	floating point type	Data type describing output values from sine and cosine functions and input values to arcsine and arccosine functions

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of System_Functions:

Name	Description
Invalid_Operand	The input value is in an improper format not accepted by the operating system
Underflow	A floating point underflow was encountered during the calculations

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Degree_Operations:

Name	Type	Description
Ada_Sin	function	Sine function handling units of degrees

3.3.6.8.9.9.9.3.10.1.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Operand	Raised if the format of the input is invalid and not accepted by the operating system
Underflow	Raised if a floating point underflow error occurs during computation

3.3.6.8.9.9.9.3.10.2 COS UNIT DESIGN (CATALOG #P787-0)

This function returns the cosine of an angle with units of degrees.

3.3.6.8.9.9.9.3.10.2.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.8.9.9.9.3.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.9.3.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Degrees	In	Angle for which a cosine is desired

3.3.6.8.9.9.9.3.10.2.4 LOCAL DATA

None.

3.3.6.8.9.9.9.3.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.3.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
function Cos (Input : Degrees) return Sin_Cos_Ratio is
begin
   return Sin_Cos_Ratio(Ada_Cos(input));
```

```
exception

   when M_Lib.ROprand    => raise Invalid_Operand;
   when M_Lib.FloUndMat => raise Underflow;

end Cos;
```

3.3.6.8.9.9.9.3.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the types required by this part and defined as generic parameters for the package specification of Degree_Operations:

Name	Type	Description
Degrees	floating point type	Data type describing units of angles
Sin_Cos_Ratio	floating point type	Data type describing output values from sine and cosine functions and input values to arcsine and arccosine functions

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of System_Functions:

Name	Description
Invalid_Operand	The input value is in an improper format not accepted by the operating system
Underflow	A floating point math underflow error has occurred

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Degree_Operations:

Name	Type	Description
Ada_Cos	function	Cosine function handling units of degrees

3.3.6.8.9.9.9.3.10.2.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Operand	Raised if the input value has an improper format which is not accepted by the operating system
Underflow	Raised if a floating point math underflow error occurs during computation

3.3.6.8.9.9.9.3.10.3 TAN UNIT DESIGN (CATALOG #P788-0)

This function returns the tangent of an angle with units of degrees.

3.3.6.8.9.9.9.3.10.3.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.8.9.9.9.3.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.9.3.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Degrees	In	Angle for which a tangent is desired

3.3.6.8.9.9.9.3.10.3.4 LOCAL DATA

None.

3.3.6.8.9.9.9.3.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.3.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
function Tan (Input : Degrees) return Tan_Ratio is
```

```
   begin

      return Tan_Ratio(Ada_Tan(Input));

   exception

      when M_Lib.ROprand   => raise Invalid_Operand;
      when M_Lib.FloOveMat => raise Overflow;

   end Tan;
```

3.3.6.8.9.9.9.3.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the types required by this part and defined as generic parameters for the package specification of Degree_Operations:

Name	Type	Description
Degrees	floating point type	Data type describing units of angles
Tan_Ratio	floating point type	Data type describing output values from tangent function and input values to arctangent function

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of System_Functions:

Name	Description
Invalid_Operand	The input value is in an improper format not accepted by the operating system
Overflow	A floating point math overflow error has occurred

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Degree_Operations:

Name	Type	Description
Ada_Tan	function	Tangent function handling units of degrees

3.3.6.8.9.9.9.3.10.3.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Operand	Raised if the input value has an improper format which is not accepted by the operating system
Overflow	Raised if a floating point overflow error is encountered during computations

3.3.6.8.9.9.9.3.10.4 ARCSIN UNIT DESIGN (CATALOG #P789-0)

This function returns the Arcsin, in units of Degrees, of an input value. The input value must not be greater than 1.0 or less than -1.0.

3.3.6.8.9.9.9.3.10.4.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.8.9.9.9.3.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.9.3.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Sin_Cos_Ratio	In	Value which an arcsine is desired

3.3.6.8.9.9.9.3.10.4.4 LOCAL DATA

None.

3.3.6.8.9.9.9.3.10.4.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.3.10.4.6 PROCESSING

The following describes the processing performed by this part:

```
function Arcsin (Input : Sin_Cos_Ratio) return Degrees is

begin

   return Ada_Arcsin(Degrees(Input));

exception

   when M_Lib.ROprand   => raise Invalid_Operand;
   when M_Lib.InvArgMat => raise Invalid_Argument;

end Arcsin;
```

3.3.6.8.9.9.9.3.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the types required by this part and defined as generic parameters for the package specification of Degree_Operations:

Name	Type	Description
Degrees	floating point type	Data type describing units of angles
Sin_Cos_Ratio	floating point type	Data type describing output values from sine and cosine functions and input values to arcsine and arccosine functions

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of System_Functions:

Name	Description
Invalid_Operand	The input value is in an improper format not accepted by the operating system
Invalid_Argument	The input value is an a range unacceptable to the function being called

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Degree_Operations:

Name	Type	Description
Ada_Arcsin	function	Arcsine function handling units of degrees

3.3.6.8.9.9.9.3.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Operand	Raised if the input value is in an improper format that is not accepted by the operating system
Invalid_Argument	Raised if the absolute value of the input greater than 1.0

3.3.6.8.9.9.9.3.10.5 ARCCOS UNIT DESIGN (CATALOG #P790-0)

This function returns the Arccos, in units of Degrees, of an input value. The absolute value of the input must not be greater than 1.0.

3.3.6.8.9.9.9.3.10.5.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.8.9.9.9.3.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.9.3.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Sin_Cos_Ratio	In	Value for which an arccosine is desired

3.3.6.8.9.9.9.3.10.5.4 LOCAL DATA

None.

3.3.6.8.9.9.9.3.10.5.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.3.10.5.6 PROCESSING

The following describes the processing performed by this part:

```
function Arccos (Input : Sin_Cos_Ratio) return Degrees is

begin

   return Ada_Arccos(Degrees(input));

exception

   when M_Lib.ROprand   => raise Invalid_Operand;
   when M_Lib.InvArgMat => raise Invalid_Argument;

end Arccos;
```

3.3.6.8.9.9.9.3.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the types required by this part and defined as generic parameters for the package specification of Degree_Operations:

Name	Type	Description
Degrees	floating point type	Data type describing units of angles
Sin_Cos_Ratio	floating point type	Data type describing output values from sine and cosine functions and input values to arcsine and arccosine functions

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of System_Functions:

Name	Description
Invalid_Operand	The input value is in an improper format not accepted by the operating system
Invalid_Argument	The input value is an a range unacceptable to the function being called

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Degree_Operations:

Name	Type	Description
Ada_Arccos	function	Arccosine function handling units of degrees

3.3.6.8.9.9.9.3.10.5.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Operand	Raised if the input value is in an improper format that is not accepted by the operating system
Invalid_Argument	Raised if the absolute value of the input is greater than 1.0

3.3.6.8.9.9.9.3.10.6 ARCTAN UNIT DESIGN (CATALOG #P791-0)

This function returns the Arctangent, in units of Degrees, of an input value.

3.3.6.8.9.9.9.3.10.6.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.8.9.9.9.3.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.9.3.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Tan_Ratio	In	Value which an arctangent is desired

3.3.6.8.9.9.9.3.10.6.4 LOCAL DATA

None.

3.3.6.8.9.9.9.3.10.6.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.3.10.6.6 PROCESSING

The following describes the processing performed by this part:

```
function Arctan (Input : Tan_Ratio) return Degrees is
begin
   return Ada_Arctan(Degrees(input));
exception
   when M_Lib.ROprand => raise Invalid_Operand;
end Arctan;
```

3.3.6.8.9.9.9.3.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the types required by this part and defined as generic parameters for the package specification of Degree_Operations:

Name	Type	Description
Degrees	floating point type	Data type describing units of angles
Tan_Ratio	floating point type	Data type describing output values from tangent function and input values to arctangent function

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of System_Functions:

Name	Description
Invalid_Operand	The input value is in an improper format not accepted by the operating system

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Degree_Operations:

Name	Type	Description
Ada_Arctan	function	Arctangent function handling units of degrees

3.3.6.8.9.9.9.3.10.6.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Operand	Raised if the input value is in an improper format which is not accepted by the operating system

3.3.6.8.9.9.9.4 SQUARE ROOT PACKAGE DESIGN (CATALOG #P792-0)

This package contains the function necessary to calculate the square root of an input value.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.9.9.4.1 REQUIREMENTS ALLOCATION

Together with the other parts in the System_Functions package, this part meets CAMP requirement R223.

3.3.6.8.9.9.9.4.2 LOCAL ENTITIES DESIGN

Packages:

The following table describes the packages maintained by this part:

Name	Type	Description
New_Math_Lib	package	Math library where functions called will have inputs and outputs of type Inputs; output values will be converted as required

3.3.6.8.9.9.9.4.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined at the package specification level of this part:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Inputs	floating point type	Data type of input values
Outputs	floating	Data type of output values

3.3.6.8.9.9.9.4.4 LOCAL DATA

None.

3.3.6.8.9.9.9.4.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.4.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials.System_Functions)
package body Square_Root is

-- ------------------------
-- --instantiated package-
-- ------------------------

   package New_Math_Lib is new Math_Lib (Real => Inputs);

   package M_Lib renames New_Math_Lib;

-- ---------------------------------
-- --functions used in this package-
-- ---------------------------------

   function Ada_Sqrt (Input : Inputs) return Inputs renames M_Lib.Sqrt;

end Square_Root;
```

3.3.6.8.9.9.9.4.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.9.9.4.8 LIMITATIONS

None.

3.3.6.8.9.9.9.4.9 LLCSC DESIGN

None.

3.3.6.8.9.9.9.4.10 UNIT DESIGN

3.3.6.8.9.9.9.4.10.1 SQRT UNIT DESIGN

This function returns the square root of an input value. The input value must be greater than or equal to 0.0.

3.3.6.8.9.9.9.4.10.1.1 REQUIREMENTS ALLOCATION

Together with the other parts in the System_Functions package, this part meets CAMP required R223.

3.3.6.8.9.9.9.4.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.9.4.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Inputs	In	Value for which a square root is desired

3.3.6.8.9.9.9.4.10.1.4 LOCAL DATA

None.

3.3.6.8.9.9.9.4.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.4.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
function SqRt (Input : Inputs) return Outputs is

begin

   return Outputs(Ada_Sqrt(input));

exception

   when M_Lib.ROprand    => raise Invalid_Operand;
   when M_Lib.SquRooNeg => raise Square_Root_Negative;

end SqRt;
```

3.3.6.8.9.9.9.4.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the types required by this part and defined as generic parameters to the Square_Root package:

Name	Type	Description
Inputs	floating point type	Data type of input values
Outputs	floating	Data type of output values

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of System_Functions:

Name	Description
Invalid_Operand	The input value is in an improper format not accepted by the operating system
Square_Root_Negative	An attempt was made to take the square root of a negative number

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Square_Root:

Name	Type	Description
Ada_Sqrt	function	Square root function

3.3.6.8.9.9.9.4.10.1.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Operand	Raised if the input value has an invalid format which is not accepted by the operating system
Square_Root_Negative	Raised if an attempt is made to take the square root of a negative value

3.3.6.8.9.9.9.5 BASE_10_LOGARITHM PACKAGE DESIGN (CATALOG #P793-0)

This package contains the functions necessary to calculate the base 10 logarithm of an input value.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.9.9.5.1 REQUIREMENTS ALLOCATION

Together with the other parts in the System_Functions package, this part meets CAMP requirement R223.

3.3.6.8.9.9.9.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.9.5.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined in this part's package specification:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Inputs	floating point type	Data type of input values
Outputs	floating	Data type of output values

3.3.6.8.9.9.9.5.4 LOCAL DATA

None.

3.3.6.8.9.9.9.5.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.5.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials.System_Functions)
package body Base_10_Logarithm is

-- ------------------------
-- --instantiated package-
-- ------------------------

   package New_Math_Lib is new Math_Lib (Real => Inputs);
```

```
   package M_Lib renames New_Math_Lib;

-- ---------------------------------
-- --functions used in this package-
-- ---------------------------------

   function Ada_Log10 (Input : Inputs) return Inputs renames M_Lib.Log10;

end Base_10_Logarithm;
```

3.3.6.8.9.9.9.5.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.9.9.5.8 LIMITATIONS

None.

3.3.6.8.9.9.9.5.9 LLCSC DESIGN

None.

3.3.6.8.9.9.9.5.10 UNIT DESIGN

3.3.6.8.9.9.9.5.10.1 LOG_10 UNIT DESIGN

This function calculates the base 10 logarithm of an input value. The input value must be greater than 0.0.

3.3.6.8.9.9.9.5.10.1.1 REQUIREMENTS ALLOCATION

Together with the other parts in the System_Functions package, this parts meets CAMP requirement R223.

3.3.6.8.9.9.9.5.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.9.5.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Inputs	In	Value for which a base 10 log is desired

3.3.6.8.9.9.9.5.10.1.4 LOCAL DATA

None.

3.3.6.8.9.9.9.5.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.5.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
function Log_10 (Input : Inputs) return Outputs is

begin

   return Outputs(Ada_Log10(input));

exception

   when M_Lib.Roprand   => raise Invalid_Operand;
   when M_Lib.LogZerNeg => raise Log_Zero_Negative;

end Log_10;
```

3.3.6.8.9.9.9.5.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the types required by this part and defined as generic parameters to the package specification of Base_10_Logarithm:

Name	Type	Description
Inputs	floating point type	Data type of input values
Outputs	floating	Data type of output values

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of System_Functions:

Name	Description
Invalid_Operand	The input value is in an improper format not accepted by the operating system
Log_Zero_Negative	An attempt was made to take a log of a zero or negative value value

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Base_10_Logarithm:

Name	Type	Description
Ada_Log10	Function	Calculates the base 10 logarithm of a value

3.3.6.8.9.9.9.5.10.1.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Operand	Raised if the format of input value is invalid and not accepted by the operating system
Log_Zero_Negative	Raised if an attempt is made to take the log of a zero or negative value

3.3.6.8.9.9.9.6 BASE_N_LOGARITHM PACKAGE DESIGN (CATALOG #P794-0)

This package contains the functions necessary to calculate the base n logarithm of an input value.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.9.9.6.1 REQUIREMENTS ALLOCATION

Together with the other parts in the System_Functions package, this part meets CAMP requirement R223.

3.3.6.8.9.9.9.6.2 LOCAL ENTITIES DESIGN

Subprograms:

This package contains a sequence of statements at the end of the package body which are executed when this package is elaborated. This code initializes the object Log10_of_Base_N.

Packages:

The following table describes the packages contained in this part:

Name	Type	Description
New_Math_Lib	package	Math library where functions called will have inputs and outputs of type Inputs; output values will be converted as required

3.3.6.8.9.9.9.6.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously described in the this part's package specification:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Inputs	floating point type	Data type of input values
Outputs	floating point type	Data type of output values

Data objects:

The following table summarizes the generic formal objects required by this part:

Name	Type	Value	Description
Base_N	POSITIVE	N/A	Determines the root of the logarithm

Subprograms:

The following table summarizes the generic formal subroutines (operators) required by this part:

Name	Left Input Type	Right Input Type	Result Type
"*"	Outputs	Outputs	Outputs
"/"	Inputs	Inputs	Outputs

3.3.6.8.9.9.9.6.4 LOCAL DATA

Data objects:

The following describes the local data maintained by this part:

Name	Type	Description
One_Over_Log10_of_Base_N	Outputs	The inverse value of the log base 10 of the input value Base_N

3.3.6.8.9.9.9.6.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.6.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials.System_Functions)
package body Base_N_Logarithm is

-- ----------------------
-- --instantiated package-
-- ----------------------

   package New_Math_Lib is new Math_Lib (Real => Inputs);

   package M_Lib renames New_Math_Lib;

-- ------------------
-- --local variables-
-- ------------------

   One_Over_Log10_of_Base_N : Inputs;

-- ---------------------------------
-- --functions used in this package-
-- ---------------------------------

   function Ada_Log10 (Input : Inputs) return Inputs renames M_Lib.Log10;


-- -------------------------------------
-- --begin package body Base_N_Logarithm
-- -------------------------------------

begin

   One_Over_Log10_of_Base_N := 1.0 / Ada_Log10(Inputs(base_n));

exception
```

```
   when M_Lib.ROprand   => raise Invalid_Operand;

end Base_N_Logarithm;
```

3.3.6.8.9.9.9.6.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.9.9.6.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Operand	Raised if the format of the value of Base_N is invalid and not accepted by the operating system

3.3.6.8.9.9.9.6.9 LLCSC DESIGN

None.

3.3.6.8.9.9.9.6.10 UNIT DESIGN

3.3.6.8.9.9.9.6.10.1 LOG_N UNIT DESIGN

This function calculates the base n logarithm of an input value. It uses the following formula to do this:

$$\text{base n log (input)} := \frac{\text{base 10 log (input)}}{\text{base 10 log (n)}}$$

3.3.6.8.9.9.9.6.10.1.1 REQUIREMENTS ALLOCATION

Together with the other parts in the System_Functions package, this parts meets CAMP requirement R223.

3.3.6.8.9.9.9.6.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.9.9.6.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Inputs	In	Value for which a base n log is desired

3.3.6.8.9.9.9.6.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
log10_of_input	Inputs	Base 10 logarithm of input value

3.3.6.8.9.9.9.6.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.9.9.6.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
   function Log_N (Input : Inputs) return Outputs is

--    ---------------------
--    --declaration section
--    ---------------------

      Log10_of_Input : Inputs;

-- ---------------------------
-- --begin function Log_N
-- ---------------------------

   begin

      Log10_of_Input := Ada_Log10(Input);
      return log10_of_input * One_Over_Log10_of_Base_N;

   exception

      when M_Lib.ROprand   => raise Invalid_Operand;
      when M_Lib.LogZerNeg => raise Log_Zero_Negative;

   end Log_N;
```

3.3.6.8.9.9.9.6.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the types required by this part and defined as generic parameters for the package specification Base_N_Logarithm:

Name	Type	Description
Inputs	floating point type	Data type of input values
Outputs	floating point type	Data type of output values

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of System_Functions:

Name	Description
Invalid_Operand	The input value is in an improper format not accepted by the operating system
Log_Zero_Negative	An attempt was made to take a log of a zero or negative value value

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in ancestral units:

Name	Type	Description
Ada_Log10	Function	Calculates the base 10 logarithm of a value

3.3.6.8.9.9.9.6.10.1.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Operand	Raised if the format of the value of Base_N is invalid and not accepted by the operating system
Log_Zero_Negative	Raised if the value of Base_N is not greater than 0

3.3.6.8.9.9.10 UNIT DESIGN

None.

3.3.6.8.9.10 CONTINUED_FRACTIONS PACKAGE DESIGN (CATALOG #P730-0)

This package contains generic functions providing Continued Fractions polynomial solutions for the tangent and arctangent functions. Provisions are made for the trigonometric functions to handle units of radians.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.10.1 REQUIREMENTS ALLOCATION

N/A

3.3.6.8.9.10.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.10.3 INPUT/OUTPUT

None.

3.3.6.8.9.10.4 LOCAL DATA

None.

3.3.6.8.9.10.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.10.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials)
package body Continued_Fractions is

   package body Continued_Radian_Operations is separate;

end Continued_Fractions;
```

3.3.6.8.9.10.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.10.8 LIMITATIONS

None.

3.3.6.8.9.10.9 LLCSC DESIGN

3.3.6.8.9.10.9.1 CONTINUED_RADIAN_OPERATIONS PACKAGE DESIGN (CATALOG #P731-0)

This package contains generic functions providing Continued Fractions polynomial solutions for the tangent and arctangent functions. Provisions are made for the trigonometric functions to handle units of radians.

The following table lists the catalog numbers for subunits contained in this part:

Name	Catalog _#
Tan_R	P732-0
Arctan_R	P733-0

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.10.9.1.1 REQUIREMENTS ALLOCATION

N/A

3.3.6.8.9.10.9.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.10.9.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Radians	Floating Point	Angle expressed radians
Tan_Ratio	Floating Point	Value of computed tangent function

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Description
Default_Term_Count	Positive	Number of terms in the calculation

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Overloaded operator to multiply radians * radians yielding a tan_ratio result.

3.3.6.8.9.10.9.1.4 LOCAL DATA

None.

3.3.6.8.9.10.9.1.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.10.9.1.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials.Continued_Fractions)
package body Continued_Radian_Operations is

-- -- Tangent functions

   function Tan_R (Input      : Radians;
                   Term_Count : Positive :=Default_Term_Count )
                                                  return Tan_Ratio is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Mod_Term      : Integer;
      Result        : Tan_Ratio;

   begin
      Mod_Term := 2 * Term_Count - 1;
      Input_Squared := Input * Input;
      Inter_Result := Input_Squared;
      Divide:
         loop
            Inter_Result := Input_Squared/(Tan_Ratio(Mod_Term) - Inter_Result);
            Mod_Term := Mod_Term - 2;
            exit when Mod_Term <= 1;
         end loop Divide;
      Result := Tan_Ratio(Input) / (1.0 - Inter_Result);
      return Result;
   end Tan_R;

-- -- Arctangent functions
```

```
   function Arctan_R (Input      : Tan_Ratio;
                      Term_Count : Positive := Default_Term_Count )
                                                    return Radians is

      Count         : Positive := Term_Count;
      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Mod_Term      : Integer;
      Result        : Radians;

   begin
      Mod_Term := 2 * Term_Count - 1;
      Input_Squared := Input * Input;
      Inter_Result := Input_Squared;
      Divide:
         loop
            Inter_Result := Input_Squared /
                            (Tan_Ratio(Mod_Term) +
                             Tan_Ratio(Count * Count) *
                             Inter_Result);
            Count := Count - 1;
            Mod_Term := Mod_Term - 2;
            exit when Mod_term <= 1;
         end loop Divide;
      Result := Radians(Input / (1.0 + Inter_Result));
      return Result;
   end Arctan_R;

end Continued_Radian_Operations;
```

3.3.6.8.9.10.9.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.10.9.1.8 LIMITATIONS

None.

3.3.6.8.9.10.9.1.9 LLCSC DESIGN

None.

3.3.6.8.9.10.9.1.10 UNIT DESIGN

None.

3.3.6.8.9.10.10 UNIT DESIGN

None.

3.3.6.8.9.11 CODY_WAITE PACKAGE DESIGN (CATALOG #P880-0)

This packages contains generic functions providing Cody Waite polynomial solutions for a set of trigonometric functions. Provisions are made for the trigonometric functions to handle units of radians or degrees.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.11.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R222.

3.3.6.8.9.11.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.11.3 INPUT/OUTPUT

None.

3.3.6.8.9.11.4 LOCAL DATA

None.

3.3.6.8.9.11.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.11.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials)
package body Cody_Waite is

   package body Cody_Natural_Log is separate;

   package body Cody_Log_Base_N is separate;

end Cody_Waite;
```

3.3.6.8.9.11.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.11.8 LIMITATIONS

None.

3.3.6.8.9.11.9 LLCSC DESIGN

3.3.6.8.9.11.9.1 CODY_NATURAL_LOG PACKAGE DESIGN (CATALOG #P881-0)

This package contains a generic package providing Cody Waite polynomial solutions for the natural logarithm function. Provisions are made for the natural log function to handle units of real. Outputs are also of type real.

The following table lists the catalog numbers for subunits contained in this part:

Name	Catalog _#
Nat_Log	P882-0

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.11.9.1.1 REQUIREMENTS ALLOCATION

N/A

3.3.6.8.9.11.9.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.11.9.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by the package:

Name	Type	Description
Inputs	Floating point	Floating point Input to natural log function.
Outputs	Floating point	Floating point Output of natural log function.

FORMAL PARAMETERS:

The following table describes the formal parameters for the functions contained in this part:

Function	Name	Type	Description
Nat_Log	Input	Inputs	Input for natural log function

3.3.6.8.9.11.9.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
C0	constant	0.70710_6781	Square root of 0.5
C1	constant	8_#0.543_#	octal constant
C2	constant	- 0.21219_4440e-3	calculation constant
A0	constant	- 64.12494_34	used in R function
A1	constant	16.38394_36	used in R function
A2	constant	- 0.78956_1129	used in R function
B0	constant	- 769.49932_1	used in R function
B1	constant	312.03222_1	used in R function
B2	constant	- 35.66797_77	used in R function
B3	constant	1.0	used in R function

3.3.6.8.9.11.9.1.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.11.9.1.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials.Cody_Waite)
package body Cody_Natural_Log is

   C0 : constant Inputs :=   0.70710_67811_86547_52440; -- SQRT(0.5)
   C1 : constant Inputs :=   8_#0.543_#;
   C2 : constant Inputs := - 0.00021_21944_40054_69058_2767;

-- --used in R function
   A0 : constant Inputs := - 64.12494_34237_45581_147;
   A1 : constant Inputs :=   16.38394_35630_21534_222;
   A2 : constant Inputs :=  - 0.78956_11288_74912_57267;
   B0 : constant Inputs := - 769.49932_10849_48797_77;
   B1 : constant Inputs :=   312.03222_09192_45328_44;
   B2 : constant Inputs :=  - 35.66797_77390_34646_171;
   B3 : constant Inputs :=     1.0000_00000_00000_0000;
```

```
function Nat_Log (Input  : Inputs) return Outputs is

   F           : Inputs;
   Inter_Result : Inputs;
   N           : INTEGER;
   Result      : Outputs;
   Sign        : Inputs;
   XN          : Inputs;
   Y           : Inputs;
   Z           : Inputs;
   Zden        : Inputs;
   Znum        : Inputs;

   function R( Z : Inputs ) return Inputs is

      W : Inputs := Z * Z;

   begin
      return Z + Z * W * (A0 + (A1 + A2 * W) * W) /
                 (B0 + (B1 + (B2 + W) * W) * W);
   end R;

   procedure Defloat( Input    : Inputs;
                      Sign     : out Inputs;
                      Mantissa : out Inputs;
                      Exponent : out INTEGER) is

      X_Norm : Inputs  := Input;
      N      : INTEGER := 0;

   begin
      Sign := 1.0;
      if X_Norm = 0.0 then
         Exponent := 0;
         Mantissa := 0.0;
         return;
      elsif X_Norm < 0.0 then
         X_Norm := - X_Norm;
         Sign := -1.0;
      end if;
      if X_Norm >= 1.0 then        -- reduce to 0.5 .. 1.0
         Coarse1:
            while X_Norm >= 1024.0 loop     -- coarse reduction
               N := N + 10;
               X_Norm := X_Norm * 0.00097_65625;   -- exact on binary machine
            end loop Coarse1;
         Fine1:
            while X_Norm >= 1.0 loop      -- fine reduction
               N := N + 1;
               X_Norm := X_Norm * 0.5;    -- exact on binary machine
            end loop Fine1;
      else
         Coarse2:
            while X_Norm < 0.00097_65625 loop   -- coarse reduction
               N := N - 10;
               X_Norm := X_Norm * 1024.0;     -- exact on binary machine
            end loop Coarse2;
```

```
            Fine2:
               while X_Norm < 0.5 loop          -- fine reduction
                  N := N - 1;
                  X_Norm := X_Norm * 2.0;       -- exact on binary machine
               end loop Fine2;
            end if;
            Exponent := N;
            Mantissa := X_Norm;
         end Defloat;

      begin
         Defloat( Input, Sign, F, N );
         Znum := F - 0.5;
         if F > C0 then
            Znum := Znum - 0.5;
            Zden := F * 0.5 + 0.5;
         else
            N := N - 1;
            Zden := Znum * 0.5 + 0.5;
         end if;
         Z := Znum / Zden;
         if N = 0 then
            Inter_Result := R( Z );
         else
            Xn := Inputs(N);
            Inter_Result := (Xn * C2 + R( Z )) + Xn * C1;
         end if;
         Result := Outputs(Inter_Result);
         return Result;
      end Nat_Log;

   end Cody_Natural_Log;
```

3.3.6.8.9.11.9.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.11.9.1.8 LIMITATIONS

None.

3.3.6.8.9.11.9.1.9 LLCSC DESIGN

None.

3.3.6.8.9.11.9.1.10 UNIT DESIGN

None.

3.3.6.8.9.11.9.2 CODY_LOG_BASE_N PACKAGE DESIGN (CATALOG #P883-0)

This packages contains generic functions providing Cody Waite polynomial solutions for the log function for base N.

The following table lists the catalog numbers for subunits contained in this part:

Name	Catalog _#
Log_Base_N	P884-0

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.11.9.2.1 REQUIREMENTS ALLOCATION

None.

3.3.6.8.9.11.9.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.11.9.2.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Inputs	Floating point	Floating point input to the function
Outputs	Floating point	Floating point output to the function

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
Base_N	Positive	defualt = 10	Base to operate in

FORMAL PARAMETERS:

The following table describes the formal parameters for the functions contained in this part:

Function	Name	Type	Description
Log_Base_N	Input	Floating	Input upon which to apply the funtion

3.3.6.8.9.11.9.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Local_Natural_Log	Instantiated package	N/A	Natural log package used in calculating log base n

3.3.6.8.9.11.9.2.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.11.9.2.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials.Cody_Waite)
package body Cody_Log_Base_N is

   package Local_Natural_Log is new Cody_Natural_Log( Inputs  => Inputs,
                                                      Outputs => Outputs);

   package body Log_Base_N is

      One_Over_Base_Log : constant Outputs := 1.0 /
                          Local_Natural_Log.Nat_Log( Inputs(Base_N) );

      function Log_N ( Input : Inputs ) return Outputs is
      begin
         return Local_Natural_Log.Nat_Log( Input ) * One_Over_Base_Log;
      end Log_N;

   end Log_Base_N;

end Cody_Log_Base_N;
```

3.3.6.8.9.11.9.2.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.11.9.2.8 LIMITATIONS

None.

3.3.6.8.9.11.9.2.9 LLCSC DESIGN

None.

3.3.6.8.9.11.9.2.10 UNIT DESIGN

None.

3.3.6.8.9.11.10 UNIT DESIGN

None.

3.3.6.8.9.12 REDUCTION_OPERATIONS PACKAGE DESIGN (CATALOG #P1080-0)

This package contains reduction functions providing reduction of the input range for sine and cosine. The sine range is from Pi to -Pi reduced to Pi/2 to -Pi/2. Cosine reduces to 0 to Pi.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.8.9.12.1 REQUIREMENTS ALLOCATION

None.

3.3.6.8.9.12.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.12.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Inputs	Floating point	The type of the input value to be reduced

Data objects:

The following table summarizes the generic formal objects required by this part:

Name	Type	Mode	Value	Description
Quarter_Cycle	Inputs	in	Pi / 2	reduction constant of one quarter of a cycle - enables input to be of radians ,semicircles, or degrees

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Inputs	in	Value to be reduced

3.3.6.8.9.12.4 LOCAL DATA

None.

3.3.6.8.9.12.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.12.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Polynomials)
package body Reduction_Operations is

end Reduction_Operations;
```

3.3.6.8.9.12.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.12.8 LIMITATIONS

None.

3.3.6.8.9.12.9 LLCSC DESIGN

None.

3.3.6.8.9.12.10 UNIT DESIGN

3.3.6.8.9.12.10.1 SINE_REDUCTION UNIT DESIGN (CATALOG #P1082-0)

This function reduces input for the sine function from a range between Pi and -Pi to a range between Pi/2 and -Pi/2.

3.3.6.8.9.12.10.1.1 REQUIREMENTS ALLOCATION

None.

3.3.6.8.9.12.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.12.10.1.3 INPUT/OUTPUT

None.

3.3.6.8.9.12.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Result	Inputs	Result of calculations

3.3.6.8.9.12.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.12.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
function Sine_Reduction( Input : Inputs ) return Inputs is
   Result           : Inputs;
begin
   if Input > Quarter_Cycle then
      Result := Half_Cycle - Input;
   elsif Input < - Quarter_Cycle then
      Result := - Half_Cycle - Input;
   else
      Result := Input;
   end if;
   return Result;
end Sine_Reduction;
```

3.3.6.8.9.12.10.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.12.10.1.8 LIMITATIONS

None.

3.3.6.8.9.12.10.2 COSINE_REDUCTION UNIT DESIGN (CATALOG #P1084-0)

This function reduces input for the cosine function from a range between Pi and -Pi to a range between 0 and Pi.

3.3.6.8.9.12.10.2.1 REQUIREMENTS ALLOCATION

None.

3.3.6.8.9.12.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.8.9.12.10.2.3 INPUT/OUTPUT

None.

3.3.6.8.9.12.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Result	Inputs	Result of calculations

3.3.6.8.9.12.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.8.9.12.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
function Cosine_Reduction( Input : Inputs ) return Inputs is
   Result : Inputs;
begin
   return ABS( Input );
end Cosine_Reduction;
```

3.3.6.8.9.12.10.2.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.8.9.12.10.2.8 LIMITATIONS

None.

3.3.6.8.10 UNIT DESIGN

None.

(This page left intentionally blank.)

```
with Math_Lib;
package body Polynomials is

   package body Chebyshev is separate;

   package body Cody_Waite is separate;

   package body Continued_Fractions is separate;

   package body Fike is separate;

   package body General_Polynomial is separate;

   package body Hart is separate;

   package body Hastings is separate;

   package body Modified_Newton_Raphson is separate;

   package body Newton_Raphson is separate;

   package body System_Functions is separate;

   package body Taylor_Series is separate;

   package body Reduction_Operations is separate;

end Polynomials;
```

```
separate (Polynomials)
package body Chebyshev is

   package body Chebyshev_Radian_Operations is separate;

   package body Chebyshev_Degree_Operations is separate;

   package body Chebyshev_Semicircle_Operations is separate;

end Chebyshev;
```

```
separate (Polynomials.Chebyshev)
package body Chebyshev_Radian_Operations is

   Sin_R_C0 : constant :=  1.34752_631;
   Sin_R_C1 : constant := -1.55659_125;
   Sin_R_C2 : constant :=  0.22275_7911;
   Sin_R_C3 : constant := -0.01419_31743;
   Sin_R_C4 : constant :=  0.00051_19072_74;
   Sin_R_B5 : constant := -0.00001_18935_046;

   function Sin_R_5term(Input : Radians) return Sin_Cos_Ratio is

      Inter_Result_4 : Real;
      Inter_Result_3 : Real;
      Inter_Result_2 : Real;
      Inter_Result_1 : Real;
      Inter_Result_0 : Real;
      Result         : Sin_Cos_Ratio;
      Y              : Real;
      Y_Squared      : Real;

   begin
      Y := Input * One_Over_Pi;          -- converts radians to semicircles
      Y_Squared := Y * Y;
      Inter_Result_0 := 4.0 * Y_Squared - 2.0;
      Inter_Result_4 := Inter_Result_0 * Sin_R_B5 + Sin_R_C4;
      Inter_Result_3 := Inter_Result_0 * Inter_Result_4 - Sin_R_B5 +
                        Sin_R_C3;
      Inter_Result_2 := Inter_Result_0 * Inter_Result_3 - Inter_Result_4 +
                        Sin_R_C2;
      Inter_Result_1 := Inter_Result_0 * Inter_Result_2 - Inter_Result_3 +
                        Sin_R_C1;
      Inter_Result_0 := Inter_Result_0 * Inter_Result_1 - Inter_Result_2 +
                        Sin_R_C0;
      Inter_Result_0 := (Inter_Result_0 - (2.0 * Y_Squared - 1.0) *
                         Inter_Result_1) * Y;
      if Inter_Result_0 > 1.0 then
         Inter_Result_0 := 1.0;
      elsif Inter_Result_0 < -1.0 then
         Inter_Result_0 := -1.0;
      end if;
      Result := Sin_Cos_Ratio(Inter_Result_0);
      return Result;
   end Sin_R_5term;

end Chebyshev_Radian_Operations;
```

```
separate (Polynomials.Chebyshev)
package body Chebyshev_Degree_Operations is

   Sin_D_C0 : constant :=  1.34752_631;
   Sin_D_C1 : constant := -1.55659_125;
   Sin_D_C2 : constant :=  0.22275_7911;
   Sin_D_C3 : constant := -0.01419_31743;
   Sin_D_C4 : constant :=  0.00051_19072_74;
   Sin_D_B5 : constant := -0.00001_18935_046;

   One_Over_180   : constant :=  0.00555555555;

   function Sin_D_5term(Input : Degrees) return Sin_Cos_Ratio is

      Inter_Result_4 : Real;
      Inter_Result_3 : Real;
      Inter_Result_2 : Real;
      Inter_Result_1 : Real;
      Inter_Result_0 : Real;
      Result         : Sin_Cos_Ratio;
      Y              : Real;
      Y_Squared      : Real;

   begin
      Y := Input * One_Over_180;       -- converts degrees to semicircles
      Y_Squared := Y * Y;
      Inter_Result_0 := 4.0 * Y_Squared - 2.0;
      Inter_Result_4 := Inter_Result_0 * Sin_D_B5 + Sin_D_C4;
      Inter_Result_3 := Inter_Result_0 * Inter_Result_4 - Sin_D_B5
                        + Sin_D_C3;
      Inter_Result_2 := Inter_Result_0 * Inter_Result_3 - Inter_Result_4
                        + Sin_D_C2;
      Inter_Result_1 := Inter_Result_0 * Inter_Result_2 - Inter_Result_3
                        + Sin_D_C1;
      Inter_Result_0 := Inter_Result_0 * Inter_Result_1 - Inter_Result_2
                        + Sin_D_C0;
      Inter_Result_0 := (Inter_Result_0 - (2.0 * Y_Squared - 1.0) *
                         Inter_Result_1) * Y;
      if Inter_Result_0 > 1.0 then
         Inter_Result_0 := 1.0;
      elsif Inter_Result_0 < -1.0 then
         Inter_Result_0 := -1.0;
      end if;
      Result := Sin_Cos_Ratio(Inter_Result_0);
      return Result;
   end Sin_D_5term;

end Chebyshev_Degree_Operations;
```

```
separate (Polynomials.Chebyshev)
package body Chebyshev_Semicircle_Operations is

   Sin_S_C0 : constant :=  1.34752_631;
   Sin_S_C1 : constant := -1.55659_125;
   Sin_S_C2 : constant :=  0.22275_7911;
   Sin_S_C3 : constant := -0.01419_31743;
   Sin_S_C4 : constant :=  0.00051_19072_74;
   Sin_S_B5 : constant := -0.00001_18935_046;

   function Sin_S_5term(Input : Semicircles) return Sin_Cos_Ratio is

      Inter_Result_4 : Real;
      Inter_Result_3 : Real;
      Inter_Result_2 : Real;
      Inter_Result_1 : Real;
      Inter_Result_0 : Real;
      Result         : Sin_Cos_Ratio;
      Y_Squared      : Real;

   begin
      Y_Squared := Input * Input;
      Inter_Result_0 := 4.0 * Y_Squared - 2.0;
      Inter_Result_4 := Inter_Result_0 * Sin_S_B5 + Sin_S_C4;
      Inter_Result_3 := Inter_Result_0 * Inter_Result_4 - Sin_S_B5 +
                        Sin_S_C3;
      Inter_Result_2 := Inter_Result_0 * Inter_Result_3 - Inter_Result_4 +
                        Sin_S_C2;
      Inter_Result_1 := Inter_Result_0 * Inter_Result_2 - Inter_Result_3 +
                        Sin_S_C1;
      Inter_Result_0 := Inter_Result_0 * Inter_Result_1 - Inter_Result_2 +
                        Sin_S_C0;
      Inter_Result_0 := (Inter_Result_0 - (2.0 * Y_Squared - 1.0) *
                        Inter_Result_1) * Real(Input);
      if Inter_Result_0 > 1.0 then
         Inter_Result_0 := 1.0;
      elsif Inter_Result_0 < -1.0 then
         Inter_Result_0 := -1.0;
      end if;
      Result := Sin_Cos_Ratio(Inter_Result_0);
      return Result;
   end Sin_S_5term;

end Chebyshev_Semicircle_Operations;
```

```
separate (Polynomials)
package body Fike is

   package body Fike_Semicircle_Operations is separate;

end Fike;
```

```
separate (Polynomials.Fike)
package body Fike_Semicircle_Operations is

   Arcsin_C1  : constant := 0.31830_9886;
   Arcsin_C3  : constant := 0.05305_20148;
   Arcsin_C5  : constant := 0.02385_63606;
   Arcsin_C7  : constant := 0.01448_96675;
   Arcsin_C9  : constant := 0.00763_75322_8;
   Arcsin_C11 : constant := 0.01350_18593;

   function Arcsin_S_6term (Input : Sin_Cos_Ratio) return Semicircles is

      Input_Squared : Real;
      Inter_Result  : Real;
      Left_Quadrant : BOOLEAN;
      Mod_Input     : Real;
      Result        : Semicircles;

   begin
      if abs( Input ) > 0.5 then
         Mod_Input := Sqrt( Real((1.0 - abs(Input)) * 0.5) );
         Left_Quadrant := TRUE;
      else
         Mod_Input := Real(Input);
         Left_Quadrant := FALSE;
      end if;
      Input_Squared := Mod_Input * Mod_Input;
      Inter_Result := (((((Arcsin_C11 * Input_Squared +
                     Arcsin_C9) * Input_Squared +
                     Arcsin_C7) * Input_Squared +
                     Arcsin_C5) * Input_Squared +
                     Arcsin_C3) * Input_Squared +
                     Arcsin_C1) * Mod_Input;
      if Left_Quadrant then
         if Input > 0.0 then
            Inter_Result := 0.5 - (2.0 * Inter_Result);
         else
            Inter_Result := - (0.5 - (2.0 * Inter_Result));
         end if;
      end if;
      Result := Semicircles(Inter_Result);
      return Result;
   end Arcsin_S_6term;

   function Arccos_S_6term (Input : Sin_Cos_Ratio) return Semicircles is

      Input_Squared : Real;
      Inter_Result  : Real;
      Left_Quadrant : BOOLEAN;
      Mod_Input     : Real;
      Result        : Semicircles;

   begin
      if abs( Input ) > 0.5 then
         Mod_Input := Sqrt( Real((1.0 - abs(Input)) * 0.5) );
         Left_Quadrant := TRUE;
      else
```

```
         Mod_Input := Real(Input);
         Left_Quadrant := FALSE;
      end if;
      Input_Squared := Mod_Input * Mod_Input;
      Inter_Result := (((((Arcsin_C11 * Input_Squared +
                    Arcsin_C9) * Input_Squared +
                    Arcsin_C7) * Input_Squared +
                    Arcsin_C5) * Input_Squared +
                    Arcsin_C3) * Input_Squared +
                    Arcsin_C1) * Mod_Input;
      if Left_Quadrant then
         if Input > 0.0 then
            Inter_Result := 0.5 - (2.0 * Inter_Result);
         else
            Inter_Result := - (0.5 - (2.0 * Inter_Result));
         end if;
      end if;
      Result := Semicircles(Inter_Result);
--    -- convert to Arccos by applying formula
--    -- Arccosine = Pi/2 - Arcsine
      Result := 0.5 - Result;
      return Result;
   end Arccos_S_6term;

end Fike_Semicircle_Operations;
```

```
separate (Polynomials)
package body Hart is

   package body Hart_Radian_Operations is separate;

   package body Hart_Degree_Operations is separate;

end Hart;
```

```
separate (Polynomials.Hart)
package body Hart_Radian_Operations is

   Cos_R_C0  : constant :=  0.99999_9953;
   Cos_R_C2  : constant := -0.49999_9053;
   Cos_R_C4  : constant :=  0.04166_35847;
   Cos_R_C6  : constant := -0.00138_53704_3;
   Cos_R_C8  : constant :=  0.00002_31539_317;

   function Cos_R_5term (Input : Radians) return Sin_Cos_Ratio is

      Inter_Result : Real;
      Mod_Input    : Radians;
      Result       : Sin_Cos_Ratio;
      X_Squared    : Radians;

   begin

      if Input >= Pi_Over_2 then
         Mod_Input := Pi - Input;
      else
         Mod_Input := Input;
      end if;
      X_Squared := Mod_Input * Mod_Input;
      Inter_Result := (((Cos_R_C8  * X_Squared +
                         Cos_R_C6) * X_Squared +
                        Cos_R_C4) * X_Squared +
                        Cos_R_C2) * X_Squared;
      Inter_Result := Inter_Result + Cos_R_C0;
      if Input >= Pi_Over_2 then
         Inter_Result := - Inter_Result;
      end if;
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;

   end Cos_R_5term;

end Hart_Radian_Operations;
```

```
separate (Polynomials.Hart)
package body Hart_Degree_Operations is

   Cos_D_C0 : constant :=   0.99999_9953;
   Cos_D_C2 : constant := - 1.52308_42e-04;
   Cos_D_C4 : constant :=   3.86603_79e-09;
   Cos_D_C6 : constant := - 3.91588_67e-14;
   Cos_D_C8 : constant :=   1.99362_60e-19;

   function Cos_D_5term (Input : Degrees) return Sin_Cos_Ratio is

      Inter_Result : Real;
      Mod_Input    : Degrees;
      Result       : Sin_Cos_Ratio;
      X_Squared    : Real;

   begin

      if Input >= 90.0 then
         Mod_Input := 180.0 - Input;
      else
         Mod_Input := Input;
      end if;
      X_Squared := Mod_Input * Mod_Input;
      Inter_Result := (((Cos_D_C8  * X_Squared +
                         Cos_D_C6) * X_Squared +
                         Cos_D_C4) * X_Squared +
                         Cos_D_C2) * X_Squared;
      Inter_Result := Inter_Result + Cos_D_C0;
      if Input >= 90.0 then
         Inter_Result := - Inter_Result;
      end if;
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;

   end Cos_D_5term;

end Hart_Degree_Operations;
```

```
separate (Polynomials)
package body Hastings is

   package body Hastings_Radian_Operations is separate;

   package body Hastings_Degree_Operations is separate;

end Hastings;
```

```
separate (Polynomials.Hastings)
package body Hastings_Radian_Operations is

   Sin_R_C1_5term : constant :=  0.99999_9995;
   Sin_R_C3_5term : constant := -0.16666_6567;
   Sin_R_C5_5term : constant :=  0.00833_30251_7;
   Sin_R_C7_5term : constant := -0.00019_80741_43;
   Sin_R_C9_5term : constant :=  0.00000_26018_8690;

   Sin_R_C1_4term : constant :=  0.99999_9;
   Sin_R_C3_4term : constant := -0.16665_5;
   Sin_R_C5_4term : constant :=  0.00831_190;
   Sin_R_C7_4term : constant := -0.00018_4882;

   Arctan_R_C1_8term  : constant :=  0.99999_9333;
   Arctan_R_C3_8term  : constant := -0.33329_8560;
   Arctan_R_C5_8term  : constant :=  0.19946_5360;
   Arctan_R_C7_8term  : constant := -0.13908_5335;
   Arctan_R_C9_8term  : constant :=  0.09642_00441;
   Arctan_R_C11_8term : constant := -0.05590_98861;
   Arctan_R_C13_8term : constant :=  0.02186_12288;
   Arctan_R_C15_8term : constant := -0.00405_40580;

   Arctan_R_C1_7term  : constant :=  0.99999_6115;
   Arctan_R_C3_7term  : constant := -0.33317_3758;
   Arctan_R_C5_7term  : constant :=  0.19807_8690;
   Arctan_R_C7_7term  : constant := -0.13233_5096;
   Arctan_R_C9_7term  : constant :=  0.07962_6318;
   Arctan_R_C11_7term : constant := -0.03360_6269;
   Arctan_R_C13_7term : constant :=  0.00681_2411;

   Arctan_R_C1_6term  : constant :=  0.99997_726;
   Arctan_R_C3_6term  : constant := -0.33262_347;
   Arctan_R_C5_6term  : constant :=  0.19354_346;
   Arctan_R_C7_6term  : constant := -0.11643_287;
   Arctan_R_C9_6term  : constant :=  0.05265_332;
   Arctan_R_C11_6term : constant := -0.01172_120;

   pragma PAGE;
-- --sine functions

   function Sin_R_5term(Input : Radians) return Sin_Cos_Ratio is

      Input_Squared : Real;
      Inter_Result  : Real;
      Result        : Sin_Cos_Ratio;

   begin

      Input_Squared := Input * Input;
      Inter_Result  :=              ((((Sin_R_C9_5term  *
                       Input_Squared + Sin_R_C7_5term) *
                       Input_Squared + Sin_R_C5_5term) *
                       Input_Squared + Sin_R_C3_5term) *
                       Input_Squared + Sin_R_C1_5term);
      Inter_Result := Inter_Result * Real(Input);
      if Inter_Result > 1.0 then
```

```
      Inter_Result := 1.0;
   elsif Inter_Result < -1.0 then
      Inter_Result := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result );
   return Result;

end Sin_R_5term;

pragma PAGE;
function Sin_R_4term(Input : Radians) return Sin_Cos_Ratio is

   Input_Squared : Real;
   Inter_Result  : Real;
   Result        : Sin_Cos_Ratio;

begin

   Input_Squared := Input * Input;
   Inter_Result   :=              (((Sin_R_C7_4term  *
                    Input_Squared + Sin_R_C5_4term) *
                    Input_Squared + Sin_R_C3_4term) *
                    Input_Squared + Sin_R_C1_4term);
   Inter_Result := Inter_Result * Real(Input);
   if Inter_Result > 1.0 then
      Inter_Result := 1.0;
   elsif Inter_Result < -1.0 then
      Inter_Result := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result );
   return Result;

end Sin_R_4term;

-- --cosine functions

pragma PAGE;
function Cos_R_5term(Input : Radians) return Sin_Cos_Ratio is

   Input_Squared : Real;
   Inter_Result  : Real;
   Mod_Input     : Radians;
   Result        : Sin_Cos_Ratio;

begin

   Mod_Input := Pi_Over_2 - Input;
   Input_Squared := Mod_Input * Mod_Input;
   Inter_Result   :=            ((((Sin_R_C9_5term  *
                    Input_Squared + Sin_R_C7_5term) *
                    Input_Squared + Sin_R_C5_5term) *
                    Input_Squared + Sin_R_C3_5term) *
                    Input_Squared + Sin_R_C1_5term);
   Inter_Result := Inter_Result * Real(Mod_Input);
   if Inter_Result > 1.0 then
      Inter_Result := 1.0;
   elsif Inter_Result < -1.0 then
```

```
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;

   end Cos_R_5term;

   pragma PAGE;
   function Cos_R_4term(Input : Radians) return Sin_Cos_Ratio is

      Input_Squared : Real;
      Inter_Result  : Real;
      Mod_Input     : Radians;
      Result        : Sin_Cos_Ratio;

   begin

      Mod_Input := Pi_Over_2 - Input;
      Input_Squared := Mod_Input * Mod_Input;
      Inter_Result   :=              (((Sin_R_C7_4term  *
                        Input_Squared + Sin_R_C5_4term) *
                        Input_Squared + Sin_R_C3_4term) *
                        Input_Squared + Sin_R_C1_4term);
      Inter_Result := Inter_Result * Real(Mod_Input);
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;

   end Cos_R_4term;

   pragma PAGE;
-- -- Tangent functions

   function Tan_R_5term (Input : Radians) return Tan_Ratio is
      Sin : Sin_Cos_Ratio;
      Cos : Sin_Cos_Ratio;
   begin
      Sin := Sin_R_5term(Input);
      if Input < 0.0 then
         Cos := - Cos_R_5term( Pi + Input );
      else
         Cos := Cos_R_5term(Input);
      end if;
      return Tan_Ratio(Sin / Cos);
   end Tan_R_5term;

   pragma PAGE;
   function Tan_R_4term (Input : Radians) return Tan_Ratio is
      Sin : Sin_Cos_Ratio;
      Cos : Sin_Cos_Ratio;
   begin
      Sin := Sin_R_4term(Input);
      if Input < 0.0 then
```

```
         Cos := - Cos_R_4term( Pi + Input );
      else
         Cos := Cos_R_4term(Input);
      end if;
      return Tan_Ratio(Sin / Cos);
   end Tan_R_4term;

   pragma PAGE;
-- -- Arctangent functions

   function Arctan_R_8term (Input : Tan_Ratio) return Radians is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Result        : Radians;

   begin
      Input_Squared := Input * Input;
      Inter_Result   :=          (((((((Arctan_R_C15_8term  *
                        Input_Squared + Arctan_R_C13_8term) *
                        Input_Squared + Arctan_R_C11_8term) *
                        Input_Squared + Arctan_R_C9_8term)  *
                        Input_Squared + Arctan_R_C7_8term)  *
                        Input_Squared + Arctan_R_C5_8term)  *
                        Input_Squared + Arctan_R_C3_8term)  *
                        Input_Squared + Arctan_R_C1_8term)  *
                        Input;
      Result := Radians(Inter_Result);
      return Result;
   end Arctan_R_8term;

   pragma PAGE;
   function Arctan_R_7term (Input : Tan_Ratio) return Radians is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Result        : Radians;

   begin
      Input_Squared := Input * Input;
      Inter_Result   :=            ((((((Arctan_R_C13_7term  *
                        Input_Squared + Arctan_R_C11_7term) *
                        Input_Squared + Arctan_R_C9_7term)  *
                        Input_Squared + Arctan_R_C7_7term)  *
                        Input_Squared + Arctan_R_C5_7term)  *
                        Input_Squared + Arctan_R_C3_7term)  *
                        Input_Squared + Arctan_R_C1_7term)  *
                        Input;
      Result := Radians(Inter_Result);
      return Result;
   end Arctan_R_7term;

   pragma PAGE;
   function Arctan_R_6term (Input : Tan_Ratio) return Radians is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
```

```
      Result        : Radians;

   begin
      Input_Squared := Input * Input;
      Inter_Result   :=            (((((Arctan_R_C11_6term  *
                        Input_Squared + Arctan_R_C9_6term)  *
                        Input_Squared + Arctan_R_C7_6term)  *
                        Input_Squared + Arctan_R_C5_6term)  *
                        Input_Squared + Arctan_R_C3_6term)  *
                        Input_Squared + Arctan_R_C1_6term)  *
                        Input;
      Result := Radians(Inter_Result);
      return Result;
   end Arctan_R_6term;

   pragma PAGE;
-- -- Modified Hastings Arctangent functions

   function Mod_Arctan_R_8term (Input : Tan_Ratio) return Radians is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Mod_Input     : Tan_Ratio;
      Result        : Radians;

   begin
      if Input >= 0.0 then
         Mod_Input := Input;
      else
         Mod_Input := - Input;
      end if;
      Mod_Input := (Mod_Input - 1.0) / (Mod_Input + 1.0);
      Input_Squared := Mod_Input * Mod_Input;
      Inter_Result   :=          ((((((Arctan_R_C15_8term  *
                        Input_Squared + Arctan_R_C13_8term) *
                        Input_Squared + Arctan_R_C11_8term) *
                        Input_Squared + Arctan_R_C9_8term)  *
                        Input_Squared + Arctan_R_C7_8term)  *
                        Input_Squared + Arctan_R_C5_8term)  *
                        Input_Squared + Arctan_R_C3_8term)  *
                        Input_Squared + Arctan_R_C1_8term)  *
                        Mod_Input;
      Result := Radians(Inter_Result) + Pi_Over_4;
      if Input < 0.0 then
         Result := - Result;
      end if;
      return Result;
   end Mod_Arctan_R_8term;

   pragma PAGE;
   function Mod_Arctan_R_7term (Input : Tan_Ratio) return Radians is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Mod_Input     : Tan_Ratio;
      Result        : Radians;
```

```
   begin
      if Input >= 0.0 then
         Mod_Input := Input;
      else
         Mod_Input := - Input;
      end if;
      Mod_Input := (Mod_Input - 1.0) / (Mod_Input + 1.0);
      Input_Squared := Mod_Input * Mod_Input;
      Inter_Result   :=           (((((Arctan_R_C13_7term  *
                        Input_Squared + Arctan_R_C11_7term) *
                        Input_Squared + Arctan_R_C9_7term)  *
                        Input_Squared + Arctan_R_C7_7term)  *
                        Input_Squared + Arctan_R_C5_7term)  *
                        Input_Squared + Arctan_R_C3_7term)  *
                        Input_Squared + Arctan_R_C1_7term)  *
                        Mod_Input;
      Result := Radians(Inter_Result) + Pi_Over_4;
      if Input < 0.0 then
         Result := - Result;
      end if;
      return Result;
   end Mod_Arctan_R_7term;

   pragma PAGE;
   function Mod_Arctan_R_6term (Input : Tan_Ratio) return Radians is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Mod_Input     : Tan_Ratio;
      Result        : Radians;

   begin
      if Input >= 0.0 then
         Mod_Input := Input;
      else
         Mod_Input := - Input;
      end if;
      Mod_Input := (Mod_Input - 1.0) / (Mod_Input + 1.0);
      Input_Squared := Mod_Input * Mod_Input;
      Inter_Result   :=           ((((Arctan_R_C11_6term  *
                        Input_Squared + Arctan_R_C9_6term)  *
                        Input_Squared + Arctan_R_C7_6term)  *
                        Input_Squared + Arctan_R_C5_6term)  *
                        Input_Squared + Arctan_R_C3_6term)  *
                        Input_Squared + Arctan_R_C1_6term)  *
                        Mod_Input;
      Result := Radians(Inter_Result) + Pi_Over_4;
      if Input < 0.0 then
         Result := - Result;
      end if;
      return Result;
   end Mod_Arctan_R_6term;


end Hastings_Radian_Operations;
```

```
separate (Polynomials.Hastings)
package body Hastings_Degree_Operations is

   Sin_D_C1_5term : constant :=   1.74532_92e-02;
   Sin_D_C3_5term : constant := - 8.86095_625e-07;
   Sin_D_C5_5term : constant :=   1.34955_172e-11;
   Sin_D_C7_5term : constant := - 9.77168_260e-17;
   Sin_D_C9_5term : constant :=   3.91006_135e-22;

   Sin_D_C1_4term : constant :=   1.74533e-02;
   Sin_D_C3_4term : constant := - 8.86037e-07;
   Sin_D_C5_4term : constant :=   1.34613e-11;
   Sin_D_C7_4term : constant := - 9.12087e-17;

-- --sine functions

   pragma PAGE;
   function Sin_D_5term(Input : Degrees) return Sin_Cos_Ratio is

      Input_Squared : Real;
      Inter_Result  : Real;
      Result        : Sin_Cos_Ratio;

   begin

      Input_Squared := Input * Input;
      Inter_Result   :=             ((((Sin_D_C9_5term  *
                        Input_Squared + Sin_D_C7_5term) *
                        Input_Squared + Sin_D_C5_5term) *
                        Input_Squared + Sin_D_C3_5term) *
                        Input_Squared + Sin_D_C1_5term);
      Inter_Result := Inter_Result * Real(Input);
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;

   end Sin_D_5term;

   pragma PAGE;
   function Sin_D_4term(Input : Degrees) return Sin_Cos_Ratio is

      Input_Squared : Real;
      Inter_Result  : Real;
      Result        : Sin_Cos_Ratio;

   begin

      Input_Squared := Input * Input;
      Inter_Result   :=             (((Sin_D_C7_4term  *
                        Input_Squared + Sin_D_C5_4term) *
                        Input_Squared + Sin_D_C3_4term) *
                        Input_Squared + Sin_D_C1_4term);
      Inter_Result := Inter_Result * Real(Input);
```

```
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;

   end Sin_D_4term;

   pragma PAGE;
-- --cosine functions

   function Cos_D_5term(Input : Degrees) return Sin_Cos_Ratio is

      Input_Squared : Real;
      Inter_Result  : Real;
      Mod_Input     : Degrees;
      Result        : Sin_Cos_Ratio;

   begin

      Mod_Input := 90.0 - Input;
      Input_Squared := Mod_Input * Mod_Input;
      Inter_Result   :=              ((((Sin_D_C9_5term  *
                        Input_Squared + Sin_D_C7_5term) *
                        Input_Squared + Sin_D_C5_5term) *
                        Input_Squared + Sin_D_C3_5term) *
                        Input_Squared + Sin_D_C1_5term);
      Inter_Result := Inter_Result * Real(Mod_Input);
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;

   end Cos_D_5term;

   pragma PAGE;
   function Cos_D_4term(Input : Degrees) return Sin_Cos_Ratio is

      Input_Squared : Real;
      Inter_Result  : Real;
      Mod_Input     : Degrees;
      Result        : Sin_Cos_Ratio;

   begin

      Mod_Input := 90.0 - Input;
      Input_Squared := Mod_Input * Mod_Input;
      Inter_Result   :=                (((Sin_D_C7_4term  *
                        Input_Squared + Sin_D_C5_4term) *
                        Input_Squared + Sin_D_C3_4term) *
                        Input_Squared + Sin_D_C1_4term);
      Inter_Result := Inter_Result * Real(Mod_Input);
```

```
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;

   end Cos_D_4term;

   pragma PAGE;
-- -- Tangent function

   function Tan_D_5term (Input : Degrees) return Tan_Ratio is
      Sin : Sin_Cos_Ratio;
      Cos : Sin_Cos_Ratio;
   begin
      Sin := Sin_D_5term(Input);
      if Input < 0.0 then
         Cos := - Cos_D_5term( 180.0 + Input );
      else
         Cos := Cos_D_5term(Input);
      end if;
      return Tan_Ratio(Sin / Cos);
   end Tan_D_5term;

   function Tan_D_4term (Input : Degrees) return Tan_Ratio is
      Sin : Sin_Cos_Ratio;
      Cos : Sin_Cos_Ratio;
   begin
      Sin := Sin_D_4term(Input);
      if Input < 0.0 then
         Cos := - Cos_D_4term( 180.0 + Input );
      else
         Cos := Cos_D_4term(Input);
      end if;
      return Tan_Ratio(Sin / Cos);
   end Tan_D_4term;

end Hastings_Degree_Operations;
```

```
separate (Polynomials)
package body Modified_Newton_Raphson is

   C1 : constant :=  2.18518_306;
   C2 : constant :=  3.02289_917;
   C3 : constant :=  1.54515_776;

   pragma PAGE;
   function Sqrt(Input : Inputs) return Outputs is

      Inter_Result : Reals;
      Result       : Outputs;
      Root_Pwr     : Reals;
      X_Norm       : Reals;

   begin
      if Input = 0.0 then
         Result := 0.0;
      else
         X_Norm   := Reals( Input );
--       ---------------------------------------
--       - Reduce input to between 0.25 and 1.0 -
--       - in order to achieve better initial   -
--       - approximation                        -
--       ---------------------------------------
         Root_Pwr := 1.0;
         if Input > 1.0 then
            Reduce:
               while X_Norm > 1.0 loop
                  Root_Pwr := Root_Pwr * 2.0;
                  X_Norm   := X_Norm * 0.25;
               end loop Reduce;
         else
            Increase:
               while X_Norm < 0.25 loop
                  Root_Pwr := Root_Pwr * 0.5;
                  X_Norm   := X_Norm * 4.0;
               end loop Increase;
         end if;
         Inter_Result := C1 - C2 / (X_Norm + C3);
         Inter_Result := (X_Norm/Inter_Result + Inter_Result) * 0.5;
         Inter_Result := (X_Norm/Inter_Result + Inter_Result) * 0.5;
         Inter_Result := (X_Norm/Inter_Result + Inter_Result) * 0.5;
         Inter_Result := Inter_Result * Root_Pwr;
         Result := Outputs(Inter_Result);
      end if;                          -- Input not 0.0
      return Result;
   end Sqrt;

end Modified_Newton_Raphson;
```

```
separate (Polynomials)
package body Newton_Raphson is

   C1 : constant :=  3.57142_857;
   C2 : constant := 14.57725_95;
   C3 : constant :=  0.30612_2449;
   C4 : constant :=  4.79591_837;
   C5 : constant :=  0.16659_7251;

   function Sqrt(Input : Inputs) return Outputs is

      Inter_Result : Reals;
      Result       : Outputs;
      Root_Pwr     : Reals;
      X_Norm       : Reals;

   begin
      if Input = 0.0 then
         Result := 0.0;
      else
         X_Norm   := Reals( Input );
--       ---------------------------------------
--       - Reduce input to between 0.25 and 1.0 -
--       - in order to achieve better initial   -
--       - approximation                        -
--       ---------------------------------------
         Root_Pwr := 1.0;
         if Input > 1.0 then
            Reduce:
               while X_Norm > 1.0 loop
                  Root_Pwr := Root_Pwr * 2.0;
                  X_Norm   := X_Norm * 0.25;
               end loop Reduce;
         else
            Increase:
               while X_Norm < 0.25 loop
                  Root_Pwr := Root_Pwr * 0.5;
                  X_Norm   := X_Norm * 4.0;
               end loop Increase;
         end if;
         Inter_Result := C1 - (C2 * (X_Norm + C3)) / ((X_Norm + C4) * (X_Norm + C5) + C5);
         Inter_Result := (X_Norm/Inter_Result + Inter_Result) * 0.5;
         Inter_Result := (X_Norm/Inter_Result + Inter_Result) * 0.5;
         Inter_Result := (X_Norm/Inter_Result + Inter_Result) * 0.5;
         Inter_Result := Inter_Result * Root_Pwr;
         Result := Outputs(Inter_Result);

      end if;
      return Result;
   end Sqrt;

end Newton_Raphson;
```

```
separate (Polynomials)
package body Taylor_Series is

   package body Taylor_Radian_Operations is separate;

   package body Taylor_Degree_Operations is separate;

   package body Taylor_Natural_Log is separate;

   package body Taylor_Log_Base_N is separate;

end Taylor_Series;
```

```
separate (Polynomials.Taylor_Series)
package body Taylor_Radian_Operations is

-- -- The Sine constants are used in the Taylor operations for computing
   -- the sine. The Cosine constants are used in computing cosines. In
   -- the Modified Taylor operations, however, both sets of constants are
   -- used. Constants are given for 9 digits of precision for both extended
   -- and single precision. They are named to correspond to the power
   -- of X (Input) with which they are termed.

   Sin_R_C3  : constant := -0.16666_6667;
   Sin_R_C5  : constant :=  0.00833_33333_3;
   Sin_R_C7  : constant := -0.00019_84126_98;
   Sin_R_C9  : constant :=  0.00000_27557_3164;
   Sin_R_C11 : constant := -0.00000_00250_51870_9;
   Sin_R_C13 : constant :=  0.00000_00001_60478_446;
   Sin_R_C15 : constant := -0.00000_00000_00737_06627_8;

   Cos_R_C3  : constant := -0.50000_0000;
   Cos_R_C5  : constant :=  0.04166_66667;
   Cos_R_C7  : constant := -0.00138_88888_9;
   Cos_R_C9  : constant :=  0.00002_48015_873;
   Cos_R_C11 : constant := -0.00000_02755_73192;
   Cos_R_C13 : constant :=  0.00000_00020_87675_70;
   Cos_R_C15 : constant := -0.00000_00000_11470_7456;

   Tan_R_C3  : constant := 0.33333_3333;
   Tan_R_C5  : constant := 0.13333_3333;
   Tan_R_C7  : constant := 0.05396_82540;
   Tan_R_C9  : constant := 0.02186_9488;
   Tan_R_C11 : constant := 0.00886_32355_3;
   Tan_R_C13 : constant := 0.00359_21280_4;
   Tan_R_C15 : constant := 0.00145_58343_9;

   Arcsin_R_C3  : constant := 0.16666_6666;
   Arcsin_R_C5  : constant := 0.075;
   Arcsin_R_C7  : constant := 0.04464_28571;
   Arcsin_R_C9  : constant := 0.03038_19444;
   Arcsin_R_C11 : constant := 0.02237_21591;
   Arcsin_R_C13 : constant := 0.01735_27644;
   Arcsin_R_C15 : constant := 0.01396_48438;

   Arctan_R_C3  : constant :=  0.33333_3333;
   Arctan_R_C5  : constant := -0.2;
   Arctan_R_C7  : constant :=  0.14285_7142;
   Arctan_R_C9  : constant := -0.11111_1111;
   Arctan_R_C11 : constant :=  0.09090_90909;
   Arctan_R_C13 : constant := -0.07692_30769;
   Arctan_R_C15 : constant :=  0.06666_66667;

   Alt_Arctan_R_C3  : constant := -0.33333_3333;
   Alt_Arctan_R_C5  : constant :=  0.2;
   Alt_Arctan_R_C7  : constant := -0.14285_7142;
   Alt_Arctan_R_C9  : constant :=  0.11111_1111;
   Alt_Arctan_R_C11 : constant := -0.09090_90909;
   Alt_Arctan_R_C13 : constant :=  0.07692_30769;
   Alt_Arctan_R_C15 : constant := -0.06666_66667;
```

```
   pragma PAGE;
-- -- Taylor Sine functions

   function Sin_R_8term (Input : Radians) return Sin_Cos_Ratio is

      Inter_Result  : Real;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Radians;

   begin
      X_Squared := Input * Input;
      Inter_Result :=  ((((((Sin_R_C15  * X_Squared +
                             Sin_R_C13) * X_Squared +
                             Sin_R_C11) * X_Squared +
                             Sin_R_C9)  * X_Squared +
                             Sin_R_C7)  * X_Squared +
                             Sin_R_C5)  * X_Squared +
                             Sin_R_C3)  * X_Squared;
      Inter_Result := Inter_Result * Real(Input) + Real(Input);
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Sin_R_8term;

   pragma PAGE;
   function Sin_R_7term (Input : Radians) return Sin_Cos_Ratio is

      Inter_Result  : Real;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Radians;

   begin
      X_Squared := Input * Input;
      Inter_Result :=  (((((Sin_R_C13  * X_Squared +
                            Sin_R_C11) * X_Squared +
                            Sin_R_C9)  * X_Squared +
                            Sin_R_C7)  * X_Squared +
                            Sin_R_C5)  * X_Squared +
                            Sin_R_C3)  * X_Squared;
      Inter_Result := Inter_Result * Real(Input) + Real(Input);
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Sin_R_7term;

   pragma PAGE;
   function Sin_R_6term (Input : Radians) return Sin_Cos_Ratio is
```

```
      Inter_Result  : Real;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Radians;

   begin
      X_Squared := Input * Input;
      Inter_Result :=  ((((Sin_R_C11 * X_Squared +
                           Sin_R_C9) * X_Squared +
                           Sin_R_C7) * X_Squared +
                           Sin_R_C5) * X_Squared +
                           Sin_R_C3) * X_Squared;
      Inter_Result := Inter_Result * Real(Input) + Real(Input);
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Sin_R_6term;

   pragma PAGE;
   function Sin_R_5term (Input : Radians) return Sin_Cos_Ratio is

      Inter_Result  : Real;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Radians;

   begin
      X_Squared := Input * Input;
      Inter_Result :=  (((Sin_R_C9  * X_Squared +
                          Sin_R_C7) * X_Squared +
                          Sin_R_C5) * X_Squared +
                          Sin_R_C3) * X_Squared;
      Inter_Result := Inter_Result * Real(Input) + Real(Input);
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Sin_R_5term;

   pragma PAGE;
   function Sin_R_4term (Input : Radians) return Sin_Cos_Ratio is

      Inter_Result  : Real;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Radians;

   begin
      X_Squared := Input * Input;
      Inter_Result :=  ((Sin_R_C7  * X_Squared +
                         Sin_R_C5) * X_Squared +
                         Sin_R_C3) * X_Squared;
      Inter_Result := Inter_Result * Real(Input) + Real(Input);
```

```
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Sin_R_4term;

   pragma PAGE;
-- -- Taylor Cosine functions

   function Cos_R_8term (Input : Radians) return Sin_Cos_Ratio is

      Inter_Result  : Real;
      Mod_Input     : Radians;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Radians;

   begin
      if Input > Pi_Over_2 then
         Mod_Input := Pi - Input;
      else
         Mod_Input := Input;
      end if;
      X_Squared := Mod_Input * Mod_Input;
      Inter_Result := ((((((Cos_R_C15  * X_Squared +
                            Cos_R_C13) * X_Squared +
                            Cos_R_C11) * X_Squared +
                            Cos_R_C9) * X_Squared +
                            Cos_R_C7) * X_Squared +
                            Cos_R_C5) * X_Squared +
                            Cos_R_C3) * X_Squared;
      Inter_Result := Inter_Result + 1.0;
      if Input > Pi_Over_2 then
         Inter_Result := - Inter_Result;
      end if;
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
   return Result;
   end Cos_R_8term;

   pragma PAGE;
   function Cos_R_7term (Input : Radians) return Sin_Cos_Ratio is

      Inter_Result  : Real;
      Mod_Input     : Radians;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Radians;

   begin
      if Input > Pi_Over_2 then
         Mod_Input := Pi - Input;
```

```
      else
         Mod_Input := Input;
      end if;
      X_Squared := Mod_Input * Mod_Input;
      Inter_Result := (((((Cos_R_C13  * X_Squared +
                           Cos_R_C11) * X_Squared +
                           Cos_R_C9) * X_Squared +
                           Cos_R_C7) * X_Squared +
                           Cos_R_C5) * X_Squared +
                           Cos_R_C3) * X_Squared;
      Inter_Result := Inter_Result + 1.0;
      if Input > Pi_Over_2 then
         Inter_Result := - Inter_Result;
      end if;
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
   return Result;
   end Cos_R_7term;

   pragma PAGE;
   function Cos_R_6term (Input : Radians) return Sin_Cos_Ratio is

      Inter_Result  : Real;
      Mod_Input     : Radians;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Radians;

   begin
      if Input > Pi_Over_2 then
         Mod_Input := Pi - Input;
      else
         Mod_Input := Input;
      end if;
      X_Squared := Mod_Input * Mod_Input;
      Inter_Result := ((((Cos_R_C11  * X_Squared +
                          Cos_R_C9) * X_Squared +
                          Cos_R_C7) * X_Squared +
                          Cos_R_C5) * X_Squared +
                          Cos_R_C3) * X_Square.;
      Inter_Result := Inter_Result + 1.0;
      if Input > Pi_Over_2 then
         Inter_Result := - Inter_Result;
      end if;
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
   return Result;
   end Cos_R_6term;

   pragma PAGE;
```

```
function Cos_R_5term (Input : Radians) return Sin_Cos_Ratio is

   Inter_Result  : Real;
   Mod_Input     : Radians;
   Result        : Sin_Cos_Ratio;
   X_Squared     : Radians;

begin
   if Input > Pi_Over_2 then
      Mod_Input := Pi - Input;
   else
      Mod_Input := Input;
   end if;
   X_Squared := Mod_Input * Mod_Input;
   Inter_Result := (((Cos_R_C9  * X_Squared +
                      Cos_R_C7) * X_Squared +
                      Cos_R_C5) * X_Squared +
                      Cos_R_C3) * X_Squared;
   Inter_Result := Inter_Result + 1.0;
   if Input > Pi_Over_2 then
      Inter_Result := - Inter_Result;
   end if;
   if Inter_Result > 1.0 then
      Inter_Result := 1.0;
   elsif Inter_Result < -1.0 then
      Inter_Result := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result );
return Result;
end Cos_R_5term;

pragma PAGE;
function Cos_R_4term (Input : Radians) return Sin_Cos_Ratio is

   Inter_Result  : Real;
   Mod_Input     : Radians;
   Result        : Sin_Cos_Ratio;
   X_Squared     : Radians;

begin
   if Input > Pi_Over_2 then
      Mod_Input := Pi - Input;
   else
      Mod_Input := Input;
   end if;
   X_Squared := Mod_Input * Mod_Input;
   Inter_Result := ((Cos_R_C7  * X_Squared +
                     Cos_R_C5) * X_Squared +
                     Cos_R_C3) * X_Squared;
   Inter_Result := Inter_Result + 1.0;
   if Input > Pi_Over_2 then
      Inter_Result := - Inter_Result;
   end if;
   if Inter_Result > 1.0 then
      Inter_Result := 1.0;
   elsif Inter_Result < -1.0 then
      Inter_Result := -1.0;
```

```
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
   return Result;
   end Cos_R_4term;

   pragma PAGE;
-- -- Taylor tangent functions

   function Tan_R_8term (Input : Radians) return Tan_Ratio is

      Inter_Result  : Real;
      Result        : Tan_Ratio;
      X_Squared     : Radians;

   begin
      X_Squared := Input * Input;
      Inter_Result :=  (((((Tan_R_C15  * X_Squared +
                            Tan_R_C13) * X_Squared +
                            Tan_R_C11) * X_Squared +
                            Tan_R_C9)  * X_Squared +
                            Tan_R_C7)  * X_Squared +
                            Tan_R_C5)  * X_Squared +
                            Tan_R_C3)  * X_Squared;
      Result := Tan_Ratio(Inter_Result * Real(Input) + Real(Input));
      return Result;
   end Tan_R_8term;

   pragma PAGE;
-- -- Taylor arcsine functions

   function Arcsin_R_8term (Input : Sin_Cos_Ratio) return Radians is

      Inter_Result  : Real;
      Result        : Radians;
      X_Squared     : Real;

   begin
      X_Squared := Real(Input * Input);
      Inter_Result :=  ((((((Arcsin_R_C15  * X_Squared +
                            Arcsin_R_C13) * X_Squared +
                            Arcsin_R_C11) * X_Squared +
                            Arcsin_R_C9)  * X_Squared +
                            Arcsin_R_C7)  * X_Squared +
                            Arcsin_R_C5)  * X_Squared +
                            Arcsin_R_C3)  * X_Squared;
      Result := Radians(Inter_Result * Real(Input) + Real(Input));
      return Result;
   end Arcsin_R_8term;

   pragma PAGE;
   function Arcsin_R_7term (Input : Sin_Cos_Ratio) return Radians is

      Inter_Result  : Real;
      Result        : Radians;
      X_Squared     : Real;

   begin
```

```
      X_Squared := Real(Input * Input);
      Inter_Result :=  (((((Arcsin_R_C13  * X_Squared +
                           Arcsin_R_C11) * X_Squared +
                           Arcsin_R_C9)  * X_Squared +
                           Arcsin_R_C7)  * X_Squared +
                           Arcsin_R_C5)  * X_Squared +
                           Arcsin_R_C3)  * X_Squared;
      Result := Radians(Inter_Result * Real(Input) + Real(Input));
      return Result;
   end Arcsin_R_7term;

   pragma PAGE;
   function Arcsin_R_6term (Input : Sin_Cos_Ratio) return Radians is

      Inter_Result  : Real;
      Result        : Radians;
      X_Squared     : Real;

   begin
      X_Squared := Real(Input * Input);
      Inter_Result :=  ((((Arcsin_R_C11  * X_Squared +
                           Arcsin_R_C9)  * X_Squared +
                           Arcsin_R_C7)  * X_Squared +
                           Arcsin_R_C5)  * X_Squared +
                           Arcsin_R_C3)  * X_Squared;
      Result := Radians(Inter_Result * Real(Input) + Real(Input));
      return Result;
   end Arcsin_R_6term;

   pragma PAGE;
   function Arcsin_R_5term (Input : Sin_Cos_Ratio) return Radians is

      Inter_Result  : Real;
      Result        : Radians;
      X_Squared     : Real;

   begin
      X_Squared := Real(Input * Input);
      Inter_Result :=  (((Arcsin_R_C9  * X_Squared +
                          Arcsin_R_C7) * X_Squared +
                          Arcsin_R_C5) * X_Squared +
                          Arcsin_R_C3) * X_Squared;
      Result := Radians(Inter_Result * Real(Input) + Real(Input));
      return Result;
   end Arcsin_R_5term;

   pragma PAGE;
-- -- Taylor arccosine functions

   function Arccos_R_8term (Input : Sin_Cos_Ratio) return Radians is

      Inter_Result  : Real;
      Result        : Radians;
      X_Squared     : Real;

   begin
      X_Squared := Real(Input * Input);
```

```
      Inter_Result :=  ((((((Arcsin_R_C15  * X_Squared +
                             Arcsin_R_C13) * X_Squared +
                             Arcsin_R_C11) * X_Squared +
                             Arcsin_R_C9)  * X_Squared +
                             Arcsin_R_C7)  * X_Squared +
                             Arcsin_R_C5)  * X_Squared +
                             Arcsin_R_C3)  * X_Squared;
      Result := Pi_Over_2 - (Radians(Inter_Result * Rea..(Input) + Real(Input)));
      return Result;
   end Arccos_R_8term;

   pragma PAGE;
   function Arccos_R_7term (Input : Sin_Cos_Ratio) return Radians is

      Inter_Result  : Real;
      Result        : Radians;
      X_Squared     : Real;

   begin
      X_Squared := Real(Input * Input);
      Inter_Result :=  (((((Arcsin_R_C13  * X_Squared +
                            Arcsin_R_C11) * X_Squared +
                            Arcsin_R_C9)  * X_Squared +
                            Arcsin_R_C7)  * X_Squared +
                            Arcsin_R_C5)  * X_Squared +
                            Arcsin_R_C3)  * X_Squared;
      Result := Pi_Over_2 - (Radians(Inter_Result * Real(Input) + Real(Input)));
      return Result;
   end Arccos_R_7term;

   pragma PAGE;
   function Arccos_R_6term (Input : Sin_Cos_Ratio) return Radians is

      Inter_Result  : Real;
      Result        : Radians;
      X_Squared     : Real;

   begin
      X_Squared := Real(Input * Input);
      Inter_Result :=  ((((Arcsin_R_C11  * X_Squared +
                           Arcsin_R_C9)  * X_Squared +
                           Arcsin_R_C7)  * X_Squared +
                           Arcsin_R_C5)  * X_Squared +
                           Arcsin_R_C3)  * X_Squared;
      Result := Pi_Over_2 - (Radians(Inter_Result * Real(Input) + Real(Input)));
      return Result;
   end Arccos_R_6term;

   pragma PAGE;
   function Arccos_R_5term (Input : Sin_Cos_Ratio) return Radians is

      Inter_Result  : Real;
      Result        : Radians;
      X_Squared     : Real;

   begin
      X_Squared := Real(Input * Input);
```

```
      Inter_Result :=  (((Arcsin_R_C9  * X_Squared +
                         Arcsin_R_C7) * X_Squared +
                         Arcsin_R_C5) * X_Squared +
                         Arcsin_R_C3) * X_Squared;
      Result := Pi_Over_2 - (Radians(Inter_Result * Real(Input) + Real(Input)));
      return Result;
   end Arccos_R_5term;

   pragma PAGE;
-- -- Taylor Arctangent functions
-- -- Used when |Input| > 1

   function Arctan_R_8term(Input : Tan_Ratio) return Radians is

      Input_Squared : Radians;
      Inverse       : Tan_Ratio;
      Result        : Radians;
      Temp          : Radians;

   begin
      if Input <= 1.0 then
         Temp := -PI_Over_2;
      else
         Temp := Pi_Over_2;
      end if;
      Inverse := 1.0 / Input;
      Input_Squared := Radians(Inverse * Inverse);
      Result := Temp + (((((((Arctan_R_C15  * Input_Squared +
                              Arctan_R_C13) * Input_Squared +
                              Arctan_R_C11) * Input_Squared +
                              Arctan_R_C9)  * Input_Squared +
                              Arctan_R_C7)  * Input_Squared +
                              Arctan_R_C5)  * Input_Squared +
                              Arctan_R_C3)  * Input_Squared -
                              1.0) * Radians(Inverse);
      return Result;
   end Arctan_R_8term;

   pragma PAGE;
   function Arctan_R_7term(Input : Tan_Ratio) return Radians is

      Input_Squared : Radians;
      Inverse       : Tan_Ratio;
      Result        : Radians;
      Temp          : Radians;

   begin
      if Input <= 1.0 then
         Temp := -PI_Over_2;
      else
         Temp := Pi_Over_2;
      end if;
      Inverse := 1.0 / Input;
      Input_Squared := Radians(Inverse * Inverse);
      Result := Temp + ((((((Arctan_R_C13  * Input_Squared +
                             Arctan_R_C11) * Input_Squared +
                             Arctan_R_C9)  * Input_Squared +
```

```
                                Arctan_R_C7)  * Input_Squared +
                                Arctan_R_C5)  * Input_Squared +
                                Arctan_R_C3)  * Input_Squared -
                                1.0) * Radians(Inverse);
      return Result;
   end Arctan_R_7term;

   pragma PAGE;
   function Arctan_R_6term(Input : Tan_Ratio) return Radians is

      Input_Squared : Radians;
      Inverse       : Tan_Ratio;
      Result        : Radians;
      Temp          : Radians;

   begin
      if Input <= 1.0 then
         Temp := -PI_Over_2;
      else
         Temp := Pi_Over_2;
      end if;
      Inverse := 1.0 / Input;
      Input_Squared := Radians(Inverse * Inverse);
      Result := Temp + (((((Arctan_R_C11 * Input_Squared +
                            Arctan_R_C9) * Input_Squared +
                            Arctan_R_C7) * Input_Squared +
                            Arctan_R_C5) * Input_Squared +
                            Arctan_R_C3) * Input_Squared -
                            1.0) * Radians(Inverse);
      return Result;
   end Arctan_R_6term;

   pragma PAGE;
   function Arctan_R_5term(Input : Tan_Ratio) return Radians is

      Input_Squared : Radians;
      Inverse       : Tan_Ratio;
      Result        : Radians;
      Temp          : Radians;

   begin
      if Input <= 1.0 then
         Temp := -PI_Over_2;
      else
         Temp := Pi_Over_2;
      end if;
      Inverse := 1.0 / Input;
      Input_Squared := Radians(Inverse * Inverse);
      Result := Temp + ((((Arctan_R_C9 * Input_Squared +
                           Arctan_R_C7) * Input_Squared +
                           Arctan_R_C5) * Input_Squared +
                           Arctan_R_C3) * Input_Squared -
                           1.0) * Radians(Inverse);
      return Result;
   end Arctan_R_5term;

   pragma PAGE;
```

```
   function Arctan_R_4term(Input : Tan_Ratio) return Radians is

      Input_Squared : Radians;
      Inverse       : Tan_Ratio;
      Result        : Radians;
      Temp          : Radians;

   begin
      if Input <= 1.0 then
         Temp := -PI_Over_2;
      else
         Temp := Pi_Over_2;
      end if;
      Inverse := 1.0 / Input;
      Input_Squared := Radians(Inverse * Inverse);
      Result := Temp + (((Arctan_R_C7  * Input_Squared +
                          Arctan_R_C5) * Input_Squared +
                          Arctan_R_C3) * Input_Squared -
                          1.0) * Radians(Inverse);
      return Result;
   end Arctan_R_4term;

   pragma PAGE;
-- -- Alternate Taylor Arctangent functions
-- -- Used when |Input| < 1

   function Alt_Arctan_R_8term(Input : Tan_Ratio) return Radians is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Result        : Radians;

   begin
      Input_Squared := Input * Input;
      Inter_Result := ((((((Alt_Arctan_R_C15  * Input_Squared +
                            Alt_Arctan_R_C13) * Input_Squared +
                            Alt_Arctan_R_C11) * Input_Squared +
                            Alt_Arctan_R_C9)  * Input_Squared +
                            Alt_Arctan_R_C7)  * Input_Squared +
                            Alt_Arctan_R_C5)  * Input_Squared +
                            Alt_Arctan_R_C3)  * Input_Squared;
      Result := Radians(Inter_Result * Input + Input);
      return Result;
   end Alt_Arctan_R_8term;

   pragma PAGE;
   function Alt_Arctan_R_7term(Input : Tan_Ratio) return Radians is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Result        : Radians;

   begin
      Input_Squared := Input * Input;
      Inter_Result := (((((Alt_Arctan_R_C13  * Input_Squared +
                           Alt_Arctan_R_C11) * Input_Squared +
                           Alt_Arctan_R_C9)  * Input_Squared +
```

```
                              Alt_Arctan_R_C7)  * Input_Squared +
                              Alt_Arctan_R_C5)  * Input_Squared +
                              Alt_Arctan_R_C3)  * Input_Squared;
      Result := Radians(Inter_Result * Input + Input);
      return Result;
   end Alt_Arctan_R_7term;

   pragma PAGE;
   function Alt_Arctan_R_6term(Input : Tan_Ratio) return Radians is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Result        : Radians;

   begin
      Input_Squared := Input * Input;
      Inter_Result := ((((Alt_Arctan_R_C11 * Input_Squared +
                          Alt_Arctan_R_C9) * Input_Squared +
                          Alt_Arctan_R_C7) * Input_Squared +
                          Alt_Arctan_R_C5) * Input_Squared +
                          Alt_Arctan_R_C3) * Input_Squared;
      Result := Radians(Inter_Result * Input + Input);
      return Result;
   end Alt_Arctan_R_6term;

   pragma PAGE;
   function Alt_Arctan_R_5term(Input : Tan_Ratio) return Radians is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Result        : Radians;

   begin
      Input_Squared := Input * Input;
      Inter_Result := (((Alt_Arctan_R_C9  * Input_Squared +
                         Alt_Arctan_R_C7) * Input_Squared +
                         Alt_Arctan_R_C5) * Input_Squared +
                         Alt_Arctan_R_C3) * Input_Squared;
      Result := Radians(Inter_Result * Input + Input);
      return Result;
   end Alt_Arctan_R_5term;

   pragma PAGE;
   function Alt_Arctan_R_4term(Input : Tan_Ratio) return Radians is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Result        : Radians;

   begin
      Input_Squared := Input * Input;
      Inter_Result := ((Alt_Arctan_R_C7  * Input_Squared +
                        Alt_Arctan_R_C5) * Input_Squared +
                        Alt_Arctan_R_C3) * Input_Squared;
      Result := Radians(Inter_Result * Input + Input);
      return Result;
   end Alt_Arctan_R_4term;
```

```
   pragma PAGE;
-- -- Modified Taylor Sine functions

   function Mod_Sin_R_8term(Input : Radians) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if abs(Input) >= Pi_Over_4 then
         Inter_Result_0 := Real(Pi_Over_2 - abs(Input));
         X_Squared := Inter_Result_0 * Inter_Result_0;
         Inter_Result_1 := ((((((Cos_R_C15  * X_Squared +
                                 Cos_R_C13) * X_Squared +
                                 Cos_R_C11) * X_Squared +
                                 Cos_R_C9)  * X_Squared +
                                 Cos_R_C7)  * X_Squared +
                                 Cos_R_C5)  * X_Squared +
                                 Cos_R_C3)  * X_Squared;
         Inter_Result_1 := Inter_Result_1 + 1.0;
         if Input <= - Pi_Over_4 then
            Inter_Result_1 := - Inter_Result_1;
         end if;
      else
         X_Squared := Input * Input;
         Inter_Result_1 := ((((((Sin_R_C15  * X_Squared +
                                 Sin_R_C13) * X_Squared +
                                 Sin_R_C11) * X_Squared +
                                 Sin_R_C9)  * X_Squared +
                                 Sin_R_C7)  * X_Squared +
                                 Sin_R_C5)  * X_Squared +
                                 Sin_R_C3)  * X_Squared;
         Inter_Result_1 := Inter_Result_1 * Real(Input) + Real(Input);
      end if;
      if Inter_Result_1 > 1.0 then
         Inter_Result_1 := 1.0;
      elsif Inter_Result_1 < -1.0 then
         Inter_Result_1 := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result_1 );
      return Result;
   end Mod_Sin_R_8term;

   pragma PAGE;
   function Mod_Sin_R_7term (Input : Radians) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if abs(Input) >= Pi_Over_4 then
         Inter_Result_0 := Real(Pi_Over_2 - abs(Input));
```

```
      X_Squared := Inter_Result_0 * Inter_Result_0;
      Inter_Result_1 := (((((Cos_R_C13  * X_Squared +
                            Cos_R_C11) * X_Squared +
                            Cos_R_C9)  * X_Squared +
                            Cos_R_C7)  * X_Squared +
                            Cos_R_C5)  * X_Squared +
                            Cos_R_C3)  * X_Squared;
      Inter_Result_1 := Inter_Result_1 + 1.0;
      if Input <= - Pi_Over_4 then
         Inter_Result_1 := - Inter_Result_1;
      end if;
   else
      X_Squared := Input * Input;
      Inter_Result_1 := (((((Sin_R_C13  * X_Squared +
                            Sin_R_C11) * X_Squared +
                            Sin_R_C9)  * X_Squared +
                            Sin_R_C7)  * X_Squared +
                            Sin_R_C5)  * X_Squared +
                            Sin_R_C3)  * X_Squared;
      Inter_Result_1 := Inter_Result_1 * Real(Input) + Real(Input);
   end if;
   if Inter_Result_1 > 1.0 then
      Inter_Result_1 := 1.0;
   elsif Inter_Result_1 < -1.0 then
      Inter_Result_1 := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result_1 );
   return Result;
end Mod_Sin_R_7term;

pragma PAGE;
function Mod_Sin_R_6term (Input : Radians) return Sin_Cos_Ratio is

   Inter_Result_0 : Real;
   Inter_Result_1 : Real;
   Result         : Sin_Cos_Ratio;
   X_Squared      : Real;

begin
   if abs(Input) >= Pi_Over_4 then
      Inter_Result_0 := Real(Pi_Over_2 - abs(Input));
      X_Squared := Inter_Result_0 * Inter_Result_0;
      Inter_Result_1 := ((((Cos_R_C11  * X_Squared +
                            Cos_R_C9) * X_Squared +
                            Cos_R_C7) * X_Squared +
                            Cos_R_C5) * X_Squared +
                            Cos_R_C3) * X_Squared;
      Inter_Result_1 := Inter_Result_1 + 1.0;
      if Input <= - Pi_Over_4 then
         Inter_Result_1 := - Inter_Result_1;
      end if;
   else
      X_Squared := Input * Input;
      Inter_Result_1 := ((((Sin_R_C11  * X_Squared +
                            Sin_R_C9) * X_Squared +
                            Sin_R_C7) * X_Squared +
                            Sin_R_C5) * X_Squared +
```

```
                                Sin_R_C3) * X_Squared;
      Inter_Result_1 := Inter_Result_1 * Real(Input) + Real(Input);
   end if;
   if Inter_Result_1 > 1.0 then
      Inter_Result_1 := 1.0;
   elsif Inter_Result_1 < -1.0 then
      Inter_Result_1 := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result_1 );
   return Result;
end Mod_Sin_R_6term;

pragma PAGE;
function Mod_Sin_R_5term (Input : Radians) return Sin_Cos_Ratio is

   Inter_Result_0 : Real;
   Inter_Result_1 : Real;
   Result         : Sin_Cos_Ratio;
   X_Squared      : Real;

begin
   if abs(Input) >= Pi_Over_4 then
      Inter_Result_0 := Real(Pi_Over_2 - abs(Input));
      X_Squared := Inter_Result_0 * Inter_Result_0;
      Inter_Result_1 := (((Cos_R_C9  * X_Squared +
                           Cos_R_C7) * X_Squared +
                           Cos_R_C5) * X_Squared +
                           Cos_R_C3) * X_Squared;
      Inter_Result_1 := Inter_Result_1 + 1.0;
      if Input <= - Pi_Over_4 then
         Inter_Result_1 := - Inter_Result_1;
      end if;
   else
      X_Squared := Input * Input;
      Inter_Result_1 := (((Sin_R_C9  * X_Squared +
                           Sin_R_C7) * X_Squared +
                           Sin_R_C5) * X_Squared +
                           Sin_R_C3) * X_Squared;
      Inter_Result_1 := Inter_Result_1 * Real(Input) + Real(Input);
   end if;
   if Inter_Result_1 > 1.0 then
      Inter_Result_1 := 1.0;
   elsif Inter_Result_1 < -1.0 then
      Inter_Result_1 := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result_1 );
   return Result;
end Mod_Sin_R_5term;

pragma PAGE;
function Mod_Sin_R_4term (Input : Radians) return Sin_Cos_Ratio is

   Inter_Result_0 : Real;
   Inter_Result_1 : Real;
   Result         : Sin_Cos_Ratio;
   X_Squared      : Real;
```

```
   begin
      if abs(Input) >= Pi_Over_4 then
         Inter_Result_0 := Real(Pi_Over_2 - abs(Input));
         X_Squared := Inter_Result_0 * Inter_Result_0;
         Inter_Result_1 := ((Cos_R_C7  * X_Squared +
                             Cos_R_C5) * X_Squared +
                             Cos_R_C3) * X_Squared;
         Inter_Result_1 := Inter_Result_1 + 1.0;
         if Input <= - Pi_Over_4 then
            Inter_Result_1 := - Inter_Result_1;
         end if;
      else
         X_Squared := Input * Input;
         Inter_Result_1 := ((Sin_R_C7  * X_Squared +
                             Sin_R_C5) * X_Squared +
                             Sin_R_C3) * X_Squared;
         Inter_Result_1 := Inter_Result_1 * Real(Input) + Real(Input);
      end if;
      if Inter_Result_1 > 1.0 then
         Inter_Result_1 := 1.0;
      elsif Inter_Result_1 < -1.0 then
         Inter_Result_1 := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result_1 );
      return Result;
   end Mod_Sin_R_4term;

   pragma PAGE;
-- -- Modified Taylor Cosine functions

   function Mod_Cos_R_8term(Input : Radians) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Mod_Input      : Radians;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if (Input <= Pi_Over_4) or (Input >= 3.0 * Pi_Over_4) then
         if Input > Pi_Over_2 then
            Mod_Input := Pi - Input;
         else
            Mod_Input := Input;
         end if;
         X_Squared := Mod_Input * Mod_Input;
         Inter_Result_1 := ((((((Cos_R_C15  * X_Squared +
                                 Cos_R_C13) * X_Squared +
                                 Cos_R_C11) * X_Squared +
                                 Cos_R_C9) * X_Squared +
                                 Cos_R_C7) * X_Squared +
                                 Cos_R_C5) * X_Squared +
                                 Cos_R_C3) * X_Squared;
         Inter_Result_1 :=  Inter_Result_1 + 1.0 ;
         if Input > Pi_Over_2 then
            Inter_Result_1 := - Inter_Result_1;
         end if;
```

```
      else
         Inter_Result_0 := Real(Pi_Over_2 - Input);
         X_Squared := Inter_Result_0 * Inter_Result_0;
         Inter_Result_1 := ((((((Sin_R_C15  * X_Squared +
                                 Sin_R_C13) * X_Squared +
                                 Sin_R_C11) * X_Squared +
                                 Sin_R_C9) * X_Squared +
                                 Sin_R_C7) * X_Squared +
                                 Sin_R_C5) * X_Squared +
                                 Sin_R_C3) * X_Squared;
         Inter_Result_1 := Inter_Result_1 * Inter_Result_0 + Inter_Result_0;
      end if;
      if Inter_Result_1 > 1.0 then
         Inter_Result_1 := 1.0;
      elsif Inter_Result_1 < -1.0 then
         Inter_Result_1 := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result_1 );
      return Result;
   end Mod_Cos_R_8term;

   pragma PAGE;
   function Mod_Cos_R_7term(Input : Radians) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Mod_Input      : Radians;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if (Input <= Pi_Over_4) or (Input >= 3.0 * Pi_Over_4) then
         if Input > Pi_Over_2 then
            Mod_Input := Pi - Input;
         else
            Mod_Input := Input;
         end if;
         X_Squared := Mod_Input * Mod_Input;
         Inter_Result_1 := (((((Cos_R_C13  * X_Squared +
                                Cos_R_C11) * X_Squared +
                                Cos_R_C9) * X_Squared +
                                Cos_R_C7) * X_Squared +
                                Cos_R_C5) * X_Squared +
                                Cos_R_C3) * X_Squared;
         Inter_Result_1 :=  Inter_Result_1 + 1.0 ;
         if Input > Pi_Over_2 then
            Inter_Result_1 := - Inter_Result_1;
         end if;
      else
         Inter_Result_0 := Real(Pi_Over_2 - Input);
         X_Squared := Inter_Result_0 * Inter_Result_0;
         Inter_Result_1 := ((((((Sin_R_C13  * X_Squared +
                                 Sin_R_C11) * X_Squared +
                                 Sin_R_C9) * X_Squared +
                                 Sin_R_C7) * X_Squared +
                                 Sin_R_C5) * X_Squared +
                                 Sin_R_C3) * X_Squared;
```

```
         Inter_Result_1 := Inter_Result_1 * Inter_Result_0 + Inter_Result_0;
      end if;
      if Inter_Result_1 > 1.0 then
         Inter_Result_1 := 1.0;
      elsif Inter_Result_1 < -1.0 then
         Inter_Result_1 := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result_1 );
      return Result;
   end Mod_Cos_R_7term;

   pragma PAGE;
   function Mod_Cos_R_6term(Input : Radians) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Mod_Input      : Radians;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if (Input <= Pi_Over_4) or (Input >= 3.0 * Pi_Over_4) then
         if Input > Pi_Over_2 then
            Mod_Input := Pi - Input;
         else
            Mod_Input := Input;
         end if;
         X_Squared := Mod_Input * Mod_Input;
         Inter_Result_1 := ((((Cos_R_C11  * X_Squared +
                               Cos_R_C9) * X_Squared +
                               Cos_R_C7) * X_Squared +
                               Cos_R_C5) * X_Squared +
                               Cos_R_C3) * X_Squared;
         Inter_Result_1 :=  Inter_Result_1 + 1.0 ;
         if Input > Pi_Over_2 then
            Inter_Result_1 := - Inter_Result_1;
         end if;
      else
         Inter_Result_0 := Real(Pi_Over_2 - Input);
         X_Squared := Inter_Result_0 * Inter_Result_0;
         Inter_Result_1 := ((((Sin_R_C11  * X_Squared +
                               Sin_R_C9) * X_Squared +
                               Sin_R_C7) * X_Squared +
                               Sin_R_C5) * X_Squared +
                               Sin_R_C3) * X_Squared;
         Inter_Result_1 := Inter_Result_1 * Inter_Result_0 + Inter_Result_0;
      end if;
      if Inter_Result_1 > 1.0 then
         Inter_Result_1 := 1.0;
      elsif Inter_Result_1 < -1.0 then
         Inter_Result_1 := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result_1 );
      return Result;
   end Mod_Cos_R_6term;

   pragma PAGE;
```

```
   function Mod_Cos_R_5term(Input : Radians) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Mod_Input      : Radians;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if (Input <= Pi_Over_4) or (Input >= 3.0 * Pi_Over_4) then
         if Input > Pi_Over_2 then
            Mod_Input := Pi - Input;
         else
            Mod_Input := Input;
         end if;
         X_Squared := Mod_Input * Mod_Input;
         Inter_Result_1 := (((Cos_R_C9  * X_Squared +
                              Cos_R_C7) * X_Squared +
                              Cos_R_C5) * X_Squared +
                              Cos_R_C3) * X_Squared;
         Inter_Result_1 :=  Inter_Result_1 + 1.0 ;
         if Input > Pi_Over_2 then
            Inter_Result_1 := - Inter_Result_1;
         end if;
      else
         Inter_Result_0 := Real(Pi_Over_2 - Input);
         X_Squared := Inter_Result_0 * Inter_Result_0;
         Inter_Result_1 := (((Sin_R_C9  * X_Squared +
                              Sin_R_C7) * X_Squared +
                              Sin_R_C5) * X_Squared +
                              Sin_R_C3) * X_Squared;
         Inter_Result_1 := Inter_Result_1 * Inter_Result_0 + Inter_Result_0;
      end if;
      if Inter_Result_1 > 1.0 then
         Inter_Result_1 := 1.0;
      elsif Inter_Result_1 < -1.0 then
         Inter_Result_1 := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result_1 );
      return Result;
   end Mod_Cos_R_5term;

   pragma PAGE;
   function Mod_Cos_R_4term(Input : Radians) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Mod_Input      : Radians;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if (Input <= Pi_Over_4) or (Input >= 3.0 * Pi_Over_4) then
         if Input > Pi_Over_2 then
            Mod_Input := Pi - Input;
         else
            Mod_Input := Input;
```

```
            end if;
            X_Squared := Mod_Input * Mod_Input;
            Inter_Result_1 := ((Cos_R_C7  * X_Squared +
                                Cos_R_C5) * X_Squared +
                                Cos_R_C3) * X_Squared;
            Inter_Result_1 :=  Inter_Result_1 + 1.0 ;
            if Input > Pi_Over_2 then
               Inter_Result_1 := - Inter_Result_1;
            end if;
         else
            Inter_Result_0 := Real(Pi_Over_2 - Input);
            X_Squared := Inter_Result_0 * Inter_Result_0;
            Inter_Result_1 := ((Sin_R_C7  * X_Squared +
                                Sin_R_C5) * X_Squared +
                                Sin_R_C3) * X_Squared;
            Inter_Result_1 := Inter_Result_1 * Inter_Result_0 + Inter_Result_0;
         end if;
         if Inter_Result_1 > 1.0 then
            Inter_Result_1 := 1.0;
         elsif Inter_Result_1 < -1.0 then
            Inter_Result_1 := -1.0;
         end if;
         Result := Sin_Cos_Ratio( Inter_Result_1 );
         return Result;
      end Mod_Cos_R_4term;

      pragma PAGE;
   -- -- Modified Taylor tangent functions

      function Mod_Tan_R_8term (Input : Radians) return Tan_Ratio is
      begin
         return Tan_Ratio(Sin_R_8term(Input)) /
                Tan_Ratio(Cos_R_8term(Input));
      end Mod_Tan_R_8term;

      pragma PAGE;
      function Mod_Tan_R_7term (Input : Radians) return Tan_Ratio is
      begin
         return Tan_Ratio(Sin_R_7term(Input)) /
                Tan_Ratio(Cos_R_7term(Input));
      end Mod_Tan_R_7term;

      pragma PAGE;
      function Mod_Tan_R_6term (Input : Radians) return Tan_Ratio is
      begin
         return Tan_Ratio(Sin_R_6term(Input)) /
                Tan_Ratio(Cos_R_6term(Input));
      end Mod_Tan_R_6term;

      pragma PAGE;
      function Mod_Tan_R_5term (Input : Radians) return Tan_Ratio is
      begin
         return Tan_Ratio(Sin_R_5term(Input)) /
                Tan_Ratio(Cos_R_5term(Input));
      end Mod_Tan_R_5term;

      pragma PAGE;
```

```
   function Mod_Tan_R_4term (Input : Radians) return Tan_Ratio is
   begin
      return Tan_Ratio(Sin_R_4term(Input)) /
             Tan_Ratio(Cos_R_4term(Input));
   end Mod_Tan_R_4term;

end Taylor_Radian_Operations;
```

```
separate (Polynomials.Taylor_Series)
package body Taylor_Degree_Operations is

-- -- The Sine constants are used in the Taylor operations for computing
   -- the sine. The Cosine constants are used in computing cosines. In
   -- the Mc.ified Taylor operations, however, both sets of constants are
   -- used. They are named to correspond to the power of X (Input) with
   -- which they are termed.

   Sin_D_C1  : constant :=  1.74532_92e-02;
   Sin_D_C3  : constant := -8.86096_158e-07;
   Sin_D_C5  : constant :=  1.34960_162e-11;
   Sin_D_C7  : constant := -9.78838_484e-17;
   Sin_D_C9  : constant :=  4.14126_699e-22;
   Sin_D_C11 : constant := -1.14680_931e-27;
   Sin_D_C13 : constant :=  2.23780_628e-33;
   Sin_D_C15 : constant := -3.13088_457e-39;

   Cos_D_C0  : constant :=  1.74532_92e-02;
   Cos_D_C2  : constant := -1.52308_710e-04;
   Cos_D_C4  : constant :=  3.86632_386e-09;
   Cos_D_C6  : constant := -3.92583_199e-14;
   Cos_D_C8  : constant :=  2.13549_430e-19;
   Cos_D_C10 : constant := -7.22787_516e-25;
   Cos_D_C12 : constant :=  1.66798_234e-30;
   Cos_D_C14 : constant := -2.79173_888e-36;

   Tan_D_C1  : constant := 1.74532_92e-02;
   Tan_D_C3  : constant := 1.77219_231e-06;
   Tan_D_C5  : constant := 2.15936_259e-10;
   Tan_D_C7  : constant := 2.66244_068e-14;
   Tan_D_C9  : constant := 3.28653_633e-18;
   Tan_D_C11 : constant := 4.05735_804e-22;
   Tan_D_C13 : constant := 5.00907_561e-26;
   Tan_D_C15 : constant := 6.18404_282e-30;

   pragma PAGE;
-- -- Taylor Sine functions

   function Sin_D_8term (Input : Degrees) return Sin_Cos_Ratio is

      Inter_Result  : Real;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Real;

   begin
      X_Squared := Input * Input;
      Inter_Result :=  ((((((Sin_D_C15  * X_Squared +
                             Sin_D_C13) * X_Squared +
                             Sin_D_C11) * X_Squared +
                             Sin_D_C9)  * X_Squared +
                             Sin_D_C7)  * X_Squared +
                             Sin_D_C5)  * X_Squared +
                             Sin_D_C3)  * X_Squared;
      Inter_Result := Inter_Result * Real(Input) +
                (Degrees(Sin_D_C1) * Input);
      if Inter_Result > 1.0 then
```

```
      Inter_Result := 1.0;
   elsif Inter_Result < -1.0 then
      Inter_Result := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result );
   return Result;
end Sin_D_8term;

pragma PAGE;
function Sin_D_7term (Input : Degrees) return Sin_Cos_Ratio is

   Inter_Result  : Real;
   Result        : Sin_Cos_Ratio;
   X_Squared     : Real;

begin
   X_Squared := Input * Input;
   Inter_Result :=  (((((Sin_D_C13  * X_Squared +
                         Sin_D_C11) * X_Squared +
                         Sin_D_C9)  * X_Squared +
                         Sin_D_C7)  * X_Squared +
                         Sin_D_C5)  * X_Squared +
                         Sin_D_C3)  * X_Squared;
   Inter_Result := Inter_Result * Real(Input) +
            (Degrees(Sin_D_C1) * Input);
   if Inter_Result > 1.0 then
      Inter_Result := 1.0;
   elsif Inter_Result < -1.0 then
      Inter_Result := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result );
   return Result;
end Sin_D_7term;

pragma PAGE;
function Sin_D_6term (Input : Degrees) return Sin_Cos_Ratio is

   Inter_Result  : Real;
   Result        : Sin_Cos_Ratio;
   X_Squared     : Real;

begin
   X_Squared := Input * Input;
   Inter_Result :=  ((((Sin_D_C11 * X_Squared +
                        Sin_D_C9) * X_Squared +
                        Sin_D_C7) * X_Squared +
                        Sin_D_C5) * X_Squared +
                        Sin_D_C3) * X_Squared;
   Inter_Result := Inter_Result * Real(Input) +
            (Degrees(Sin_D_C1) * Input);
   if Inter_Result > 1.0 then
      Inter_Result := 1.0;
   elsif Inter_Result < -1.0 then
      Inter_Result := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result );
   return Result;
```

```
   end Sin_D_6term;

   pragma PAGE;
   function Sin_D_5term (Input : Degrees) return Sin_Cos_Ratio is

      Inter_Result  : Real;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Real;

   begin
      X_Squared := Input * Input;
      Inter_Result :=  (((Sin_D_C9  * X_Squared +
                          Sin_D_C7) * X_Squared +
                          Sin_D_C5) * X_Squared +
                          Sin_D_C3) * X_Squared;
      Inter_Result := Inter_Result * Real(Input) +
                (Degrees(Sin_D_C1) * Input);
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Sin_D_5term;

   pragma PAGE;
   function Sin_D_4term (Input : Degrees) return Sin_Cos_Ratio is

      Inter_Result  : Real;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Real;

   begin
      X_Squared := Input * Input;
      Inter_Result :=  ((Sin_D_C7  * X_Squared +
                         Sin_D_C5) * X_Squared +
                         Sin_D_C3) * X_Squared;
      Inter_Result := Inter_Result * Real(Input) +
                (Degrees(Sin_D_C1) * Input);
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Sin_D_4term;

   pragma PAGE;
-- -- Taylor Cosine functions

   function Cos_D_8term (Input : Degrees) return Sin_Cos_Ratio is

      Inter_Result  : Real;
      Mod_Input     : Degrees;
      Result        : Sin_Cos_Ratio;
```

```
      X_Squared     : Degrees;

   begin
      if Input > 90.0 then
         Mod_Input := 180.0 - Input;
      else
         Mod_Input := Input;
      end if;
      X_Squared := Mod_Input * Mod_Input;
      Inter_Result := ((((((Cos_D_C14  * X_Squared +
                            Cos_D_C12) * X_Squared +
                            Cos_D_C10) * X_Squared +
                            Cos_D_C8) * X_Squared +
                            Cos_D_C6) * X_Squared +
                            Cos_D_C4) * X_Squared +
                            Cos_D_C2) * X_Squared;
      Inter_Result := Inter_Result + 1.0;
      if Input > 90.0 then
         Inter_Result := - Inter_Result;
      end if;
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Cos_D_8term;

   pragma PAGE;
   function Cos_D_7term (Input : Degrees) return Sin_Cos_Ratio is

      Inter_Result  : Real;
      Mod_Input     : Degrees;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Degrees;

   begin
      if Input > 90.0 then
         Mod_Input := 180.0 - Input;
      else
         Mod_Input := Input;
      end if;
      X_Squared := Mod_Input * Mod_Input;
      Inter_Result := (((((Cos_D_C12  * X_Squared +
                           Cos_D_C10) * X_Squared +
                           Cos_D_C8) * X_Squared +
                           Cos_D_C6) * X_Squared +
                           Cos_D_C4) * X_Squared +
                           Cos_D_C2) * X_Squared;
      Inter_Result := Inter_Result + 1.0;
      if Input > 90.0 then
         Inter_Result := - Inter_Result;
      end if;
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
```

```
      Inter_Result := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result );
   return Result;
end Cos_D_7term;

pragma PAGE;
function Cos_D_6term (Input : Degrees) return Sin_Cos_Ratio is

   Inter_Result  : Real;
   Mod_Input     : Degrees;
   Result        : Sin_Cos_Ratio;
   X_Squared     : Degrees;

begin
   if Input > 90.0 then
      Mod_Input := 180.0 - Input;
   else
      Mod_Input := Input;
   end if;
   X_Squared := Mod_Input * Mod_Input;
   Inter_Result := ((((Cos_D_C10  * X_Squared +
                       Cos_D_C8) * X_Squared +
                       Cos_D_C6) * X_Squared +
                       Cos_D_C4) * X_Squared +
                       Cos_D_C2) * X_Squared;
   Inter_Result := Inter_Result + 1.0;
   if Input > 90.0 then
      Inter_Result := - Inter_Result;
   end if;
   if Inter_Result > 1.0 then
      Inter_Result := 1.0;
   elsif Inter_Result < -1.0 then
      Inter_Result := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result );
   return Result;
end Cos_D_6term;

pragma PAGE;
function Cos_D_5term (Input : Degrees) return Sin_Cos_Ratio is

   Inter_Result  : Real;
   Mod_Input     : Degrees;
   Result        : Sin_Cos_Ratio;
   X_Squared     : Degrees;

begin
   if Input > 90.0 then
      Mod_Input := 180.0 - Input;
   else
      Mod_Input := Input;
   end if;
   X_Squared := Mod_Input * Mod_Input;
   Inter_Result := (((Cos_D_C8  * X_Squared +
                      Cos_D_C6) * X_Squared +
                      Cos_D_C4) * X_Squared +
```

```
                        Cos_D_C2) * X_Squared;
      Inter_Result := Inter_Result + 1.0;
      if Input > 90.0 then
         Inter_Result := - Inter_Result;
      end if;
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Cos_D_5term;

   pragma PAGE;
   function Cos_D_4term (Input : Degrees) return Sin_Cos_Ratio is

      Inter_Result  : Real;
      Mod_Input     : Degrees;
      Result        : Sin_Cos_Ratio;
      X_Squared     : Degrees;

   begin
      if Input > 90.0 then
         Mod_Input := 180.0 - Input;
      else
         Mod_Input := Input;
      end if;
      X_Squared := Mod_Input * Mod_Input;
      Inter_Result := ((Cos_D_C6  * X_Squared +
                        Cos_D_C4) * X_Squared +
                        Cos_D_C2) * X_Squared;
      Inter_Result := Inter_Result + 1.0;
      if Input > 90.0 then
         Inter_Result := - Inter_Result;
      end if;
      if Inter_Result > 1.0 then
         Inter_Result := 1.0;
      elsif Inter_Result < -1.0 then
         Inter_Result := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result );
      return Result;
   end Cos_D_4term;


   pragma PAGE;
-- -- Taylor Tangent fuctions

   function Tan_D_8term (Input : Degrees) return Tan_Ratio is

      Inter_Result  : Real;
      Result        : Tan_Ratio;
      X_Squared     : Real;

   begin
      X_Squared := Input * Input;
```

```
      Inter_Result := ((((((Tan_D_C15  * X_Squared +
                            Tan_D_C13) * X_Squared +
                            Tan_D_C11) * X_Squared +
                            Tan_D_C9)  * X_Squared +
                            Tan_D_C7)  * X_Squared +
                            Tan_D_C5)  * X_Squared +
                            Tan_D_C3)  * X_Squared;
      Result := Tan_Ratio(Inter_Result) * Tan_Ratio(Input) +
                Tan_Ratio(Input) * Tan_D_C1;
      return Result;
   end Tan_D_8term;


   pragma PAGE;
-- -- Modified Taylor Sine functions

   function Mod_Sin_D_8term(Input : Degrees) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if abs(Input) >= 45.0 then
         Inter_Result_0 := Real(90.0 - abs(Input));
         X_Squared := Inter_Result_0 * Inter_Result_0;
         Inter_Result_1 := ((((((Cos_D_C14  * X_Squared +
                                 Cos_D_C12) * X_Squared +
                                 Cos_D_C10) * X_Squared +
                                 Cos_D_C8)  * X_Squared +
                                 Cos_D_C6)  * X_Squared +
                                 Cos_D_C4)  * X_Squared +
                                 Cos_D_C2)  * X_Squared;
         Inter_Result_1 := Inter_Result_1 + 1.0;
         if Input <= - 45.0 then
            Inter_Result_1 := - Inter_Result_1;
         end if;
      else
         X_Squared := Input * Input;
         Inter_Result_1 := ((((((Sin_D_C15  * X_Squared +
                                 Sin_D_C13) * X_Squared +
                                 Sin_D_C11) * X_Squared +
                                 Sin_D_C9)  * X_Squared +
                                 Sin_D_C7)  * X_Squared +
                                 Sin_D_C5)  * X_Squared +
                                 Sin_D_C3)  * X_Squared;
         Inter_Result_1 := Inter_Result_1 * Real(Input) +
                (Degrees(Sin_D_C1) * Input);
      end if;
      if Inter_Result_1 > 1.0 then
         Inter_Result_1 := 1.0;
      elsif Inter_Result_1 < -1.0 then
         Inter_Result_1 := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result_1);
      return Result;
```

```
   end Mod_Sin_D_8term;

   pragma PAGE;
   function Mod_Sin_D_7term (Input : Degrees) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if abs(Input) >= 45.0 then
         Inter_Result_0 := Real(90.0 - abs(Input));
         X_Squared := Inter_Result_0 * Inter_Result_0;
         Inter_Result_1 := (((((Cos_D_C12  * X_Squared +
                               Cos_D_C10) * X_Squared +
                               Cos_D_C8)  * X_Squared +
                               Cos_D_C6)  * X_Squared +
                               Cos_D_C4)  * X_Squared +
                               Cos_D_C2)  * X_Squared;
         Inter_Result_1 := Inter_Result_1 + 1.0;
         if Input <= - 45.0 then
            Inter_Result_1 := - Inter_Result_1;
         end if;
      else
         X_Squared := Input * Input;
         Inter_Result_1 := (((((Sin_D_C13  * X_Squared +
                               Sin_D_C11) * X_Squared +
                               Sin_D_C9)  * X_Squared +
                               Sin_D_C7)  * X_Squared +
                               Sin_D_C5)  * X_Squared +
                               Sin_D_C3)  * X_Squared;
         Inter_Result_1 := Inter_Result_1 * Real(Input) +
                (Degrees(Sin_D_C1) * Input);
      end if;
      if Inter_Result_1 > 1.0 then
         Inter_Result_1 := 1.0;
      elsif Inter_Result_1 < -1.0 then
         Inter_Result_1 := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result_1 );
      return Result;
   end Mod_Sin_D_7term;

   pragma PAGE;
   function Mod_Sin_D_6term (Input : Degrees) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if abs(Input) >= 45.0 then
         Inter_Result_0 := Real(90.0 - abs(Input));
         X_Squared := Inter_Result_0 * Inter_Result_0;
         Inter_Result_1 := ((((Cos_D_C10  * X_Squared +
```

```
                                Cos_D_C8) * X_Squared +
                                Cos_D_C6) * X_Squared +
                                Cos_D_C4) * X_Squared +
                                Cos_D_C2) * X_Squared;
            Inter_Result_1 := Inter_Result_1 + 1.0;
            if Input <= - 45.0 then
               Inter_Result_1 := - Inter_Result_1;
            end if;
         else
            X_Squared := Input * Input;
            Inter_Result_1 := ((((Sin_D_C11  * X_Squared +
                                Sin_D_C9) * X_Squared +
                                Sin_D_C7) * X_Squared +
                                Sin_D_C5) * X_Squared +
                                Sin_D_C3) * X_Squared;
            Inter_Result_1 := Inter_Result_1 * Real(Input) +
                  (Degrees(Sin_D_C1) * Input);
         end if;
         if Inter_Result_1 > 1.0 then
            Inter_Result_1 := 1.0;
         elsif Inter_Result_1 < -1.0 then
            Inter_Result_1 := -1.0;
         end if;
         Result := Sin_Cos_Ratio( Inter_Result_1 );
         return Result;
      end Mod_Sin_D_6term;

      pragma PAGE;
      function Mod_Sin_D_5term (Input : Degrees) return Sin_Cos_Ratio is

         Inter_Result_0 : Real;
         Inter_Result_1 : Real;
         Result         : Sin_Cos_Ratio;
         X_Squared      : Real;

      begin
         if abs(Input) >= 45.0 then
            Inter_Result_0 := Real(90.0 - abs(Input));
            X_Squared := Inter_Result_0 * Inter_Result_0;
            Inter_Result_1 := (((Cos_D_C8  * X_Squared +
                                Cos_D_C6) * X_Squared +
                                Cos_D_C4) * X_Squared +
                                Cos_D_C2) * X_Squared;
            Inter_Result_1 := Inter_Result_1 + 1.0;
            if Input <= - 45.0 then
               Inter_Result_1 := - Inter_Result_1;
            end if;
         else
            X_Squared := Input * Input;
            Inter_Result_1 := (((Sin_D_C9  * X_Squared +
                                Sin_D_C7) * X_Squared +
                                Sin_D_C5) * X_Squared +
                                Sin_D_C3) * X_Squared;
            Inter_Result_1 := Inter_Result_1 * Real(Input) +
                  (Degrees(Sin_D_C1) * Input);
         end if;
         if Inter_Result_1 > 1.0 then
```

```
      Inter_Result_1 := 1.0;
   elsif Inter_Result_1 < -1.0 then
      Inter_Result_1 := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result_1 );
   return Result;
end Mod_Sin_D_5term;

pragma PAGE;
function Mod_Sin_D_4term (Input : Degrees) return Sin_Cos_Ratio is

   Inter_Result_0 : Real;
   Inter_Result_1 : Real;
   Result         : Sin_Cos_Ratio;
   X_Squared      : Real;

begin
   if abs(Input) >= 45.0 then
      Inter_Result_0 := Real(90.0 - abs(Input));
      X_Squared := Inter_Result_0 * Inter_Result_0;
      Inter_Result_1 := ((Cos_D_C6  * X_Squared +
                          Cos_D_C4) * X_Squared +
                          Cos_D_C2) * X_Squared;
      Inter_Result_1 := Inter_Result_1 + 1.0;
      if Input <= - 45.0 then
         Inter_Result_1 := - Inter_Result_1;
      end if;
   else
      X_Squared := Input * Input;
      Inter_Result_1 := ((Sin_D_C7  * X_Squared +
                          Sin_D_C5) * X_Squared +
                          Sin_D_C3) * X_Squared;
      Inter_Result_1 := Inter_Result_1 * Real(Input) +
            (Degrees(Sin_D_C1) * Input);
   end if;
   if Inter_Result_1 > 1.0 then
      Inter_Result_1 := 1.0;
   elsif Inter_Result_1 < -1.0 then
      Inter_Result_1 := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result_1 );
   return Result;
end Mod_Sin_D_4term;

pragma PAGE;
-- -- Modified Taylor Cosine functions

function Mod_Cos_D_8term(Input : Degrees) return Sin_Cos_Ratio is

   Inter_Result_0 : Real;
   Inter_Result_1 : Real;
   Mod_Input      : Degrees;
   Result         : Sin_Cos_Ratio;
   X_Squared      : Real;

begin
   if (Input <= 45.0) or (Input >= 135.0) then
```

```
         if Input > 90.0 then
            Mod_Input := 180.0 - Input;
         else
            Mod_Input := Input;
         end if;
         X_Squared := Mod_Input * Mod_Input;
         Inter_Result_1 := ((((((Cos_D_C14  * X_Squared +
                                 Cos_D_C12) * X_Squared +
                                 Cos_D_C10) * X_Squared +
                                 Cos_D_C8) * X_Squared +
                                 Cos_D_C6) * X_Squared +
                                 Cos_D_C4) * X_Squared +
                                 Cos_D_C2) * X_Squared;
         Inter_Result_1 :=  Inter_Result_1 + 1.0 ;
         if Input > 90.0 then
            Inter_Result_1 := - Inter_Result_1;
         end if;
      else
         Inter_Result_0 := Real(90.0 - Input);
         X_Squared := Inter_Result_0 * Inter_Result_0;
         Inter_Result_1 := ((((((Sin_D_C15  * X_Squared +
                                 Sin_D_C13) * X_Squared +
                                 Sin_D_C11) * X_Squared +
                                 Sin_D_C9) * X_Squared +
                                 Sin_D_C7) * X_Squared +
                                 Sin_D_C5) * X_Squared +
                                 Sin_D_C3) * X_Squared;
         Inter_Result_1 := Inter_Result_1 * Inter_Result_0 +
               (Sin_D_C1 * Inter_Result_0);
      end if;
      if Inter_Result_1 > 1.0 then
         Inter_Result_1 := 1.0;
      elsif Inter_Result_1 < -1.0 then
         Inter_Result_1 := -1.0;
      end if;
      Result := Sin_Cos_Ratio( Inter_Result_1 );
      return Result;
   end Mod_Cos_D_8term;

   pragma PAGE;
   function Mod_Cos_D_7term(Input : Degrees) return Sin_Cos_Ratio is

      Inter_Result_0 : Real;
      Inter_Result_1 : Real;
      Mod_Input      : Degrees;
      Result         : Sin_Cos_Ratio;
      X_Squared      : Real;

   begin
      if (Input <= 45.0) or (Input >= 135.0) then
         if Input > 90.0 then
            Mod_Input := 180.0 - Input;
         else
            Mod_Input := Input;
         end if;
         X_Squared := Mod_Input * Mod_Input;
         Inter_Result_1 := (((((Cos_D_C12  * X_Squared +
```

```
                                Cos_D_C10) * X_Squared +
                                Cos_D_C8) * X_Squared +
                                Cos_D_C6) * X_Squared +
                                Cos_D_C4) * X_Squared +
                                Cos_D_C2) * X_Squared;
            Inter_Result_1 :=  Inter_Result_1 + 1.0 ;
            if Input > 90.0 then
               Inter_Result_1 := - Inter_Result_1;
            end if;
         else
            Inter_Result_0 := Real(90.0 - Input);
            X_Squared := Inter_Result_0 * Inter_Result_0;
            Inter_Result_1 := (((((Sin_D_C13  * X_Squared +
                                Sin_D_C11) * X_Squared +
                                Sin_D_C9) * X_Squared +
                                Sin_D_C7) * X_Squared +
                                Sin_D_C5) * X_Squared +
                                Sin_D_C3) * X_Squared;
            Inter_Result_1 := Inter_Result_1 * Inter_Result_0 +
                  (Sin_D_C1 * Inter_Result_0);
         end if;
         if Inter_Result_1 > 1.0 then
            Inter_Result_1 := 1.0;
         elsif Inter_Result_1 < -1.0 then
            Inter_Result_1 := -1.0;
         end if;
         Result := Sin_Cos_Ratio( Inter_Result_1 );
         return Result;
      end Mod_Cos_D_7term;

      pragma PAGE;
      function Mod_Cos_D_6term(Input : Degrees) return Sin_Cos_Ratio is

         Inter_Result_0 : Real;
         Inter_Result_1 : Real;
         Mod_Input      : Degrees;
         Result         : Sin_Cos_Ratio;
         X_Squared      : Real;

      begin
         if (Input <= 45.0) or (Input >= 135.0) then
            if Input > 90.0 then
               Mod_Input := 180.0 - Input;
            else
               Mod_Input := Input;
            end if;
            X_Squared := Mod_Input * Mod_Input;
            Inter_Result_1 := ((((Cos_D_C10  * X_Squared +
                                Cos_D_C8) * X_Squared +
                                Cos_D_C6) * X_Squared +
                                Cos_D_C4) * X_Squared +
                                Cos_D_C2) * X_Squared;
            Inter_Result_1 :=  Inter_Result_1 + 1.0 ;
            if Input > 90.0 then
               Inter_Result_1 := - Inter_Result_1;
            end if;
         else
```

```
      Inter_Result_0 := Real(90.0 - Input);
      X_Squared := Inter_Result_0 * Inter_Result_0;
      Inter_Result_1 := ((((Sin_D_C11  * X_Squared +
                            Sin_D_C9) * X_Squared +
                            Sin_D_C7) * X_Squared +
                            Sin_D_C5) * X_Squared +
                            Sin_D_C3) * X_Squared;
      Inter_Result_1 := Inter_Result_1 * Inter_Result_0 +
            (Sin_D_C1 * Inter_Result_0);
   end if;
   if Inter_Result_1 > 1.0 then
      Inter_Result_1 := 1.0;
   elsif Inter_Result_1 < -1.0 then
      Inter_Result_1 := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result_1 );
   return Result;
end Mod_Cos_D_6term;

pragma PAGE;
function Mod_Cos_D_5term(Input : Degrees) return Sin_Cos_Ratio is

   Inter_Result_0 : Real;
   Inter_Result_1 : Real;
   Mod_Input      : Degrees;
   Result         : Sin_Cos_Ratio;
   X_Squared      : Real;

begin
   if (Input <= 45.0) or (Input >= 135.0) then
      if Input > 90.0 then
         Mod_Input := 180.0 - Input;
      else
         Mod_Input := Input;
      end if;
      X_Squared := Mod_Input * Mod_Input;
      Inter_Result_1 := (((Cos_D_C8  * X_Squared +
                           Cos_D_C6) * X_Squared +
                           Cos_D_C4) * X_Squared +
                           Cos_D_C2) * X_Squared;
      Inter_Result_1 :=  Inter_Result_1 + 1.0 ;
      if Input > 90.0 then
         Inter_Result_1 := - Inter_Result_1;
      end if;
   else
      Inter_Result_0 := Real(90.0 - Input);
      X_Squared := Inter_Result_0 * Inter_Result_0;
      Inter_Result_1 := (((Sin_D_C9  * X_Squared +
                           Sin_D_C7) * X_Squared +
                           Sin_D_C5) * X_Squared +
                           Sin_D_C3) * X_Squared;
      Inter_Result_1 := Inter_Result_1 * Inter_Result_0 +
            (Sin_D_C1 * Inter_Result_0);
   end if;
   if Inter_Result_1 > 1.0 then
      Inter_Result_1 := 1.0;
   elsif Inter_Result_1 < -1.0 then
```

```
      Inter_Result_1 := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result_1 );
   return Result;
end Mod_Cos_D_5term;

pragma PAGE;
function Mod_Cos_D_4term(Input : Degrees) return Sin_Cos_Ratio is

   Inter_Result_0 : Real;
   Inter_Result_1 : Real;
   Mod_Input      : Degrees;
   Result         : Sin_Cos_Ratio;
   X_Squared      : Real;

begin
   if (Input <= 45.0) or (Input >= 135.0) then
      if Input > 90.0 then
         Mod_Input := 180.0 - Input;
      else
         Mod_Input := Input;
      end if;
      X_Squared := Mod_Input * Mod_Input;
      Inter_Result_1 := ((Cos_D_C6  * X_Squared +
                          Cos_D_C4) * X_Squared +
                          Cos_D_C2) * X_Squared;
      Inter_Result_1 :=  Inter_Result_1 + 1.0 ;
      if Input > 90.0 then
         Inter_Result_1 := - Inter_Result_1;
      end if;
   else
      Inter_Result_0 := Real(90.0 - Input);
      X_Squared := Inter_Result_0 * Inter_Result_0;
      Inter_Result_1 := ((Sin_D_C7  * X_Squared +
                          Sin_D_C5) * X_Squared +
                          Sin_D_C3) * X_Squared;
      Inter_Result_1 := Inter_Result_1 * Inter_Result_0 +
            (Sin_D_C1 * Inter_Result_0);
   end if;
   if Inter_Result_1 > 1.0 then
      Inter_Result_1 := 1.0;
   elsif Inter_Result_1 < -1.0 then
      Inter_Result_1 := -1.0;
   end if;
   Result := Sin_Cos_Ratio( Inter_Result_1 );
   return Result;
end Mod_Cos_D_4term;


pragma PAGE;
-- -- Modified Taylor Tangent functions

function Mod_Tan_D_8term (Input : Degrees) return Tan_Ratio is
begin
   return Tan_Ratio(Sin_D_8term(Input)) /
          Tan_Ratio(Cos_D_8term(Input));
end Mod_Tan_D_8term;
```

```
   pragma PAGE;
   function Mod_Tan_D_7term (Input : Degrees) return Tan_Ratio is
   begin
      return Tan_Ratio(Sin_D_7term(Input)) /
             Tan_Ratio(Cos_D_7term(Input));
   end Mod_Tan_D_7term;

   pragma PAGE;
   function Mod_Tan_D_6term (Input : Degrees) return Tan_Ratio is
   begin
      return Tan_Ratio(Sin_D_6term(Input)) /
             Tan_Ratio(Cos_D_6term(Input));
   end Mod_Tan_D_6term;

   pragma PAGE;
   function Mod_Tan_D_5term (Input : Degrees) return Tan_Ratio is
   begin
      return Tan_Ratio(Sin_D_5term(Input)) /
             Tan_Ratio(Cos_D_5term(Input));
   end Mod_Tan_D_5term;

   pragma PAGE;
   function Mod_Tan_D_4term (Input : Degrees) return Tan_Ratio is
   begin
      return Tan_Ratio(Sin_D_4term(Input)) /
             Tan_Ratio(Cos_D_4term(Input));
   end Mod_Tan_D_4term;

end Taylor_Degree_Operations;
```

```
separate (Polynomials.Taylor_Series)
package body Taylor_Natural_Log is

   Nat_Log_C1  : constant := 2.0;
   Nat_Log_C3  : constant := 0.66666_6666;
   Nat_Log_C5  : constant := 0.4;
   Nat_Log_C7  : constant := 0.28574_1428;
   Nat_Log_C9  : constant := 0.22222_2222;
   Nat_Log_C11 : constant := 0.18181_8182;
   Nat_Log_C13 : constant := 0.15384_6153;
   Nat_Log_C15 : constant := 0.13333_3333;

   pragma PAGE;
   function Nat_Log_8term ( Input : Inputs ) return Outputs is

      Inter_Result  : Inputs;
      Result        : Outputs;
      Mod_Input     : Inputs;
      Mod_Squared   : Inputs;

   begin
      Mod_Input := (Input - 1.0)/(Input + 1.0);
      Mod_Squared := Mod_Input * Mod_Input;
      Inter_Result :=  (((((Nat_Log_C15  * Mod_Squared +
                            Nat_Log_C13) * Mod_Squared +
                            Nat_Log_C11) * Mod_Squared +
                            Nat_Log_C9)  * Mod_Squared +
                            Nat_Log_C7)  * Mod_Squared +
                            Nat_Log_C5)  * Mod_Squared +
                            Nat_Log_C3)  * Mod_Squared;
      Result := (Inter_Result * Mod_Input) + (Mod_Input * Nat_Log_C1);
      return Result;
   end Nat_Log_8term;

   pragma PAGE;
   function Nat_Log_7term ( Input : Inputs ) return Outputs is

      Inter_Result  : Inputs;
      Result        : Outputs;
      Mod_Input     : Inputs;
      Mod_Squared   : Inputs;

   begin
      Mod_Input := (Input - 1.0)/(Input + 1.0);
      Mod_Squared := Mod_Input * Mod_Input;
      Inter_Result :=  ((((Nat_Log_C13  * Mod_Squared +
                           Nat_Log_C11) * Mod_Squared +
                           Nat_Log_C9)  * Mod_Squared +
                           Nat_Log_C7)  * Mod_Squared +
                           Nat_Log_C5)  * Mod_Squared +
                           Nat_Log_C3)  * Mod_Squared;
      Result := (Inter_Result * Mod_Input) + (Mod_Input * Nat_Log_C1);
      return Result;
   end Nat_Log_7term;

   pragma PAGE;
   function Nat_Log_6term ( Input : Inputs ) return Outputs is
```

```
      Inter_Result  : Inputs;
      Result        : Outputs;
      Mod_Input     : Inputs;
      Mod_Squared   : Inputs;

   begin
      Mod_Input := (Input - 1.0)/(Input + 1.0);
      Mod_Squared := Mod_Input * Mod_Input;
      Inter_Result :=  ((((Nat_Log_C11  * Mod_Squared +
                           Nat_Log_C9)  * Mod_Squared +
                           Nat_Log_C7)  * Mod_Squared +
                           Nat_Log_C5)  * Mod_Squared +
                           Nat_Log_C3)  * Mod_Squared;
      Result := (Inter_Result * Mod_Input) + (Mod_Input * Nat_Log_C1);
      return Result;
   end Nat_Log_6term;

   pragma PAGE;
   function Nat_Log_5term ( Input : Inputs ) return Outputs is

      Inter_Result  : Inputs;
      Result        : Outputs;
      Mod_Input     : Inputs;
      Mod_Squared   : Inputs;

   begin
      Mod_Input := (Input - 1.0)/(Input + 1.0);
      Mod_Squared := Mod_Input * Mod_Input;
      Inter_Result :=  (((Nat_Log_C9  * Mod_Squared +
                          Nat_Log_C7) * Mod_Squared +
                          Nat_Log_C5) * Mod_Squared +
                          Nat_Log_C3) * Mod_Squared;
      Result := (Inter_Result * Mod_Input) + (Mod_Input * Nat_Log_C1);
      return Result;
   end Nat_Log_5term;

   pragma PAGE;
   function Nat_Log_4term ( Input : Inputs ) return Outputs is

      Inter_Result  : Inputs;
      Result        : Outputs;
      Mod_Input     : Inputs;
      Mod_Squared   : Inputs;

   begin
      Mod_Input := (Input - 1.0)/(Input + 1.0);
      Mod_Squared := Mod_Input * Mod_Input;
      Inter_Result :=  ((Nat_Log_C7  * Mod_Squared +
                         Nat_Log_C5) * Mod_Squared +
                         Nat_Log_C3) * Mod_Squared;
      Result := (Inter_Result * Mod_Input) + (Mod_Input * Nat_Log_C1);
      return Result;
   end Nat_Log_4term;

end Taylor_Natural_Log;
```

```
separate (Polynomials.Taylor_Series)
package body Taylor_Log_Base_N is

   package Local_Natural_Log is new Taylor_Natural_Log( Inputs  => Inputs,
                                                        Outputs => Outputs);

   package body Log_Base_N_8term is

      One_Over_Base_Log : constant Outputs := 1.0 /
                          Local_Natural_Log.Nat_Log_8term( Inputs(Base_N) );

      function Log_N_8term ( Input : Inputs ) return Outputs is
      begin
         return Local_Natural_Log.Nat_Log_8term( Input ) * One_Over_Base_Log;
      end Log_N_8term;

   end Log_Base_N_8term;

   package body Log_Base_N_7term is

      One_Over_Base_Log : constant Outputs := 1.0 /
                          Local_Natural_Log.Nat_Log_7term( Inputs(Base_N) );

      function Log_N_7term ( Input : Inputs ) return Outputs is
      begin
            return Local_Natural_Log.Nat_Log_7term( Input ) * One_Over_Base_Log;
      end Log_N_7term;

   end Log_Base_N_7term;

   package body Log_Base_N_6term is

      One_Over_Base_Log : constant Outputs := 1.0 /
                          Local_Natural_Log.Nat_Log_6term( Inputs(Base_N) );

      function Log_N_6term ( Input : Inputs ) return Outputs is
      begin
            return Local_Natural_Log.Nat_Log_6term( Input ) * One_Over_Base_Log;
      end Log_N_6term;

   end Log_Base_N_6term;

   package body Log_Base_N_5term is

      One_Over_Base_Log : constant Outputs := 1.0 /
                          Local_Natural_Log.Nat_Log_5term( Inputs(Base_N) );

      function Log_N_5term ( Input : Inputs ) return Outputs is
      begin
            return Local_Natural_Log.Nat_Log_5term( Input ) * One_Over_Base_Log;
      end Log_N_5term;

   end Log_Base_N_5term;

   package body Log_Base_N_4term is

      One_Over_Base_Log : constant Outputs := 1.0 /
```

```
                        Local_Natural_Log.Nat_Log_4term( Inputs(Base_N) );

      function Log_N_4term ( Input : Inputs ) return Outputs is
      begin
            return Local_Natural_Log.Nat_Log_4term( Input ) * One_Over_Base_Log;
      end Log_N_4term;

   end Log_Base_N_4term;

end Taylor_Log_Base_N;
```

```
separate (Polynomials)
package body General_Polynomial is

   function Polynomial (Input : Inputs) return Results is
      Result : Results;
   begin
      Result := 0.0;
      for INDEX in Table_Dimension
         loop
            Result := Result +
                      Polynomial_Definition( INDEX ).Coefficient *
                     (Input ** Polynomial_Definition( INDEX ).Power_Of_X);
         end loop;
      return Result;
   end Polynomial;

end General_Polynomial;
```

```
separate (Polynomials)
package body System_Functions is

   package body Radian_Operations is separate;

   package body Semicircle_Operations is separate;

   package body Degree_Operations is separate;

   package body Square_Root is separate;

   package body Base_10_Logarithm is separate;

   package body Base_N_Logarithm is separate;

end System_Functions;
```

```
separate (Polynomials.System_Functions)
package body Radian_Operations is

-- ------------------------
-- --instantiated packages-
-- ------------------------

   package Radian_Math_Lib is new Math_Lib (Real => Radians);

   package M_Lib renames Radian_Math_Lib;

-- ----------------------------------------
-- --renamed functions within Local_Math_Lib
-- ----------------------------------------

   function Ada_Sin (Input : Radians) return Radians renames M_Lib.Sin;
   function Ada_Cos (Input : Radians) return Radians renames M_Lib.Cos;
   function Ada_Tan (Input : Radians) return Radians renames M_Lib.Tan;
   function Ada_Arcsin
               (Input : Radians) return Radians renames M_Lib.Asin;
   function Ada_Arccos
               (Input : Radians) return Radians renames M_Lib.Acos;
   function Ada_Arctan
               (Input : Radians) return Radians renames M_Lib.Atan;

pragma PAGE;
   function Sin (Input : Radians) return Sin_Cos_Ratio is

   begin

      return Sin_Cos_Ratio(Ada_Sin(Input));

   exception

      when M_Lib.Roprand => raise Invalid_Operand;

   end Sin;

pragma PAGE;
   function Cos (Input : Radians) return Sin_Cos_Ratio is

   begin

      return Sin_Cos_Ratio(Ada_Cos(Input));

   exception

      when M_Lib.Roprand => raise Invalid_Operand;

   end Cos;

pragma PAGE;
   function Tan (Input : Radians) return Tan_Ratio is

   begin

      return Tan_Ratio(Ada_Tan(Input));
```

```
      exception

         when M_Lib.Roprand    => raise Invalid_Operand;
         when M_Lib.Floovemat => raise Overflow;

      end Tan;

   pragma PAGE;
      function Arcsin (Input : Sin_Cos_Ratio) return Radians is

      begin

         return Ada_Arcsin(Radians(Input));

      exception

         when M_Lib.Roprand    => raise Invalid_Operand;
         when M_Lib.Invargmat => raise Invalid_Argument;

      end Arcsin;

   pragma PAGE;
      function Arccos (Input : Sin_Cos_Ratio) return Radians is

      begin

         return Ada_Arccos(Radians(Input));

      exception

         when M_Lib.Roprand    => raise Invalid_Operand;
         when M_Lib.Invargmat => raise Invalid_Argument;

      end Arccos;

   pragma PAGE;
      function Arctan (Input : Tan_Ratio) return Radians is

      begin

         return Ada_Arctan(Radians(Input));

      exception

         when M_Lib.Roprand => raise Invalid_Operand;

      end Arctan;

   end Radian_Operations;
```

```
separate (Polynomials.System_Functions)
package body Semicircle_Operations is

-- ------------------------
-- --instantiated packages-
-- ------------------------

   package Semicircle_Math_Lib is new Math_Lib (Real => Scalars);

   package M_Lib renames Semicircle_Math_Lib;

-- ---------------------------------------
-- --renamed functions within Local_Math_Lib
-- ---------------------------------------

   function Ada_Sin    (Input : Scalars) return Scalars renames M_Lib.Sin;
   function Ada_Cos    (Input : Scalars) return Scalars renames M_Lib.Cos;
   function Ada_Tan    (Input : Scalars) return Scalars renames M_Lib.Tan;
   function Ada_Arcsin
               (Input : Scalars) return Scalars renames M_Lib.Asin;
   function Ada_Arccos
               (Input : Scalars) return Scalars renames M_Lib.Acos;
   function Ada_Arctan
               (Input : Scalars) return Scalars renames M_Lib.Atan;

-- --------------------
-- --local declarations
-- --------------------

   One_Over_Pi : constant Scalars := 1.0 / Pi;

pragma PAGE;
   function Sin (Input : Semicircles) return Sin_Cos_Ratio is

   begin

      return Sin_Cos_Ratio(Ada_Sin(Input*Pi));

   exception

      when M_Lib.Roprand => raise Invalid_Operand;

   end Sin;

pragma PAGE;
   function Cos (Input : Semicircles) return Sin_Cos_Ratio is

   begin

      return Sin_Cos_Ratio(Ada_Cos(Input*Pi));

   exception

      when M_Lib.Roprand => raise Invalid_Operand;

   end Cos;
```

```
pragma PAGE;
   function Tan (Input : Semicircles) return Tan_Ratio is

   begin

      return Tan_Ratio(Ada_Tan(Input*Pi));

   exception

      when M_Lib.Roprand    => raise Invalid_Operand;
      when M_Lib.Floovemat => raise Overflow;

   end Tan;

pragma PAGE;
   function Arcsin (Input : Sin_Cos_Ratio) return Semicircles is

   begin

      return Ada_Arcsin(Scalars(Input)) * One_Over_Pi;

   exception

      when M_Lib.Roprand    => raise Invalid_Operand;
      when M_Lib.Invargmat => raise Invalid_Argument;

   end Arcsin;

pragma PAGE;
   function Arccos (Input : Sin_Cos_Ratio) return Semicircles is

   begin

      return Ada_Arccos(Scalars(Input)) * One_Over_Pi;

   exception

      when M_Lib.Roprand    => raise Invalid_Operand;
      when M_Lib.Invargmat => raise Invalid_Argument;

   end Arccos;

pragma PAGE;
   function Arctan (Input : Tan_Ratio) return Semicircles is

   begin

      return Ada_Arctan(Scalars(Input)) * One_Over_Pi;

   exception

      when M_Lib.Roprand => raise Invalid_Operand;

   end Arctan;

end Semicircle_Operations;
```

```
separate (Polynomials.System_Functions)
package body Degree_Operations is

-- ----------------------
-- --instantiated package-
-- ----------------------

   package Degree_Math_Lib is new Math_Lib (Real => Degrees);

   package M_Lib renames Degree_Math_Lib;

-- ----------------------------------------
-- --renamed functions within Degree_Math_Lib
-- ----------------------------------------

   function Ada_Sin    (Input : Degrees)
                       return Degrees renames M_Lib.Sind;
   function Ada_Cos    (Input : Degrees)
                       return Degrees renames M_Lib.Cosd;
   function Ada_Tan    (Input : Degrees)
                       return Degrees renames M_Lib.Tand;
   function Ada_Arcsin (Input : Degrees)
                       return Degrees renames M_Lib.Asind;
   function Ada_Arccos (Input : Degrees)
                       return Degrees renames M_Lib.Acosd;
   function Ada_Arctan (Input : Degrees)
                       return Degrees renames M_Lib.Atand;

pragma PAGE;
   function Sin (Input : Degrees) return Sin_Cos_Ratio is

   begin

      return Sin_Cos_Ratio(Ada_Sin(Input));

   exception

      when M_Lib.Roprand   => raise Invalid_Operand;
      when M_Lib.Floundmat => raise Underflow;

   end Sin;

pragma PAGE;
   function Cos (Input : Degrees) return Sin_Cos_Ratio is

   begin

      return Sin_Cos_Ratio(Ada_Cos(Input));

   exception

      when M_Lib.Roprand   => raise Invalid_Operand;
      when M_Lib.Floundmat => raise Underflow;

   end Cos;

pragma PAGE;
```

```
      function Tan (Input : Degrees) return Tan_Ratio is

      begin

         return Tan_Ratio(Ada_Tan(Input));

      exception

         when M_Lib.Roprand    => raise Invalid_Operand;
         when M_Lib.Floovemat => raise Overflow;

      end Tan;

   pragma PAGE;
      function Arcsin (Input : Sin_Cos_Ratio) return Degrees is

      begin

         return Ada_Arcsin(Degrees(Input));

      exception

         when M_Lib.Roprand    => raise Invalid_Operand;
         when M_Lib.Invargmat => raise Invalid_Argument;

      end Arcsin;

   pragma PAGE;
      function Arccos (Input : Sin_Cos_Ratio) return Degrees is

      begin

         return Ada_Arccos(Degrees(Input));

      exception

         when M_Lib.Roprand    => raise Invalid_Operand;
         when M_Lib.Invargmat => raise Invalid_Argument;

      end Arccos;

   pragma PAGE;
      function Arctan (Input : Tan_Ratio) return Degrees is

      begin

         return Ada_Arctan(Degrees(Input));

      exception

         when M_Lib.Roprand => raise Invalid_Operand;

      end Arctan;

   end Degree_Operations;
```

```
separate (Polynomials.System_Functions)
package body Square_Root is

-- ----------------------
-- --instantiated package-
-- ----------------------

   package New_Math_Lib is new Math_Lib (Real => Inputs);

   package M_Lib renames New_Math_Lib;

-- ---------------------------------
-- --functions used in this package-
-- ---------------------------------

   function Ada_Sqrt (Input : Inputs) return Inputs renames M_Lib.Sqrt;

pragma PAGE;
   function Sqrt (Input : Inputs) return Outputs is

   begin

      return Outputs(Ada_Sqrt(Input));

   exception

      when M_Lib.Roprand   => raise Invalid_Operand;
      when M_Lib.Squrooneg => raise Square_Root_Negative;

   end Sqrt;

end Square_Root;
```

```
separate (Polynomials.System_Functions)
package body Base_10_Logarithm is

-- ------------------------
-- --instantiated package-
-- ------------------------

   package New_Math_Lib is new Math_Lib (Real => Inputs);

   package M_Lib renames New_Math_Lib;

-- ---------------------------------
-- --functions used in this package-
-- ---------------------------------

   function Ada_Log10 (Input : Inputs) return Inputs renames M_Lib.Log10;

pragma PAGE;
   function Log_10 (Input : Inputs) return Outputs is

   begin

      return Outputs(Ada_Log10(Input));

   exception

      when M_Lib.Roprand   => raise Invalid_Operand;
      when M_Lib.Logzerneg => raise Log_Zero_Negative;

   end Log_10;

end Base_10_Logarithm;
```

```
separate (Polynomials.System_Functions)
package body Base_N_Logarithm is

-- ------------------------
-- --instantiated package-
-- ------------------------

   package New_Math_Lib is new Math_Lib (Real => Inputs);

   package M_Lib renames New_Math_Lib;

-- -------------------
-- --local variables-
-- -------------------

   One_Over_Log10_Of_Base_N : Inputs;

-- ---------------------------------
-- --functions used in this package-
-- ---------------------------------

   function Ada_Log10 (Input : Inputs) return Inputs renames M_Lib.Log10;

pragma PAGE;
   function Log_N (Input : Inputs) return Outputs is

--    ----------------------
--    --declaration section
--    ----------------------

      Log10_Of_Input : Inputs;

-- ----------------------------
-- --begin function Log_N
-- ----------------------------

   begin

      Log10_Of_Input := Ada_Log10(Input);
      return Log10_Of_Input * One_Over_Log10_Of_Base_N;

   exception

      when M_Lib.Roprand   => raise Invalid_Operand;
      when M_Lib.Logzerneg => raise Log_Zero_Negative;

   end Log_N;

pragma PAGE;
-- ------------------------------------
-- --begin package body Base_N_Logarithm
-- ------------------------------------

begin

   One_Over_Log10_Of_Base_N := 1.0 / Ada_Log10(Inputs(Base_N));
```

```
exception

   when M_Lib.Roprand    => raise Invalid_Operand;

end Base_N_Logarithm;
```

```
separate (Polynomials)
package body Continued_Fractions is

   package body Continued_Radian_Operations is separate;

end Continued_Fractions;
```

```
separate (Polynomials.Continued_Fractions)
package body Continued_Radian_Operations is

-- -- Tangent functions

   function Tan_R (Input      : Radians;
                   Term_Count : POSITIVE :=Default_Term_Count )
                                                  return Tan_Ratio is

      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Mod_Term      : INTEGER;
      Result        : Tan_Ratio;

   begin
      Mod_Term := 2 * Term_Count - 1;
      Input_Squared := Input * Input;
      Inter_Result := Input_Squared;
      Divide:
         loop
            Inter_Result := Input_Squared/(Tan_Ratio(Mod_Term) - Inter_Result);
            Mod_Term := Mod_Term - 2;
            exit when Mod_Term <= 1;
         end loop Divide;
      Result := Tan_Ratio(Input) / (1.0 - Inter_Result);
      return Result;
   end Tan_R;

-- -- Arctangent functions

   function Arctan_R (Input      : Tan_Ratio;
                      Term_Count : POSITIVE := Default_Term_Count )
                                                    return Radians is

      COUNT         : POSITIVE := Term_Count;
      Input_Squared : Tan_Ratio;
      Inter_Result  : Tan_Ratio;
      Mod_Term      : INTEGER;
      Result        : Radians;

   begin
      Mod_Term := 2 * Term_Count - 1;
      Input_Squared := Input * Input;
      Inter_Result := Input_Squared;
      Divide:
         loop
            Inter_Result := Input_Squared /
                            (Tan_Ratio(Mod_Term) +
                             Tan_Ratio(COUNT * COUNT) *
                             Inter_Result);
            COUNT := COUNT - 1;
            Mod_Term := Mod_Term - 2;
            exit when Mod_Term <= 1;
         end loop Divide;
      Result := Radians(Input / (1.0 + Inter_Result));
      return Result;
```

```
   end Arctan_R;

end Continued_Radian_Operations;
```

```
separate (Polynomials)
package body Cody_Waite is

   package body Cody_Natural_Log is separate;

   package body Cody_Log_Base_N is separate;

end Cody_Waite;
```

```
separate (Polynomials.Cody_Waite)
package body Cody_Natural_Log is

   C0 : constant Inputs :=   0.70710_67811_86547_52440; -- SQRT(0.5)
   C1 : constant Inputs :=   8_#0.543_#;
   C2 : constant Inputs := - 0.00021_21944_40054_69058_2767;

-- --used in R function
   A0 : constant Inputs := - 64.12494_34237_45581_147;
   A1 : constant Inputs :=   16.38394_35630_21534_222;
   A2 : constant Inputs :=  - 0.78956_11288_74912_57267;
   B0 : constant Inputs := - 769.49932_10849_48797_77;
   B1 : constant Inputs :=   312.03222_09192_45328_44;
   B2 : constant Inputs :=  - 35.66797_77390_34646_171;
   B3 : constant Inputs :=     1.0000_00000_00000_0000;

   function Nat_Log (Input  : Inputs) return Outputs is

      F           : Inputs;
      Inter_Result : Inputs;
      N           : INTEGER;
      Result      : Outputs;
      Sign        : Inputs;
      Xn          : Inputs;
      Y           : Inputs;
      Z           : Inputs;
      Zden        : Inputs;
      Znum        : Inputs;

      function R( Z : Inputs ) return Inputs is

         W : Inputs := Z * Z;

      begin
         return Z + Z * W * (A0 + (A1 + A2 * W) * W) /
                    (B0 + (B1 + (B2 + W) * W) * W);
      end R;

      procedure Defloat( Input    : Inputs;
                         Sign     : out Inputs;
                         MANTISSA : out Inputs;
                         Exponent : out INTEGER) is

         X_Norm : Inputs  := Input;
         N      : INTEGER := 0;

      begin
         Sign := 1.0;
         if X_Norm = 0.0 then
            Exponent := 0;
            MANTISSA := 0.0;
            return;
         elsif X_Norm < 0.0 then
            X_Norm := - X_Norm;
            Sign := -1.0;
         end if;
         if X_Norm >= 1.0 then        -- reduce to 0.5 .. 1.0
```

```
            Coarse1:
               while X_Norm >= 1024.0 loop      -- coarse reduction
                  N := N + 10;
                  X_Norm := X_Norm * 0.00097_65625;    -- exact on binary machine
               end loop Coarse1;
            Fine1:
               while X_Norm >= 1.0 loop         -- fine reduction
                  N := N + 1;
                  X_Norm := X_Norm * 0.5;      -- exact on binary machine
               end loop Fine1;
         else
            Coarse2:
               while X_Norm < 0.00097_65625 loop    -- coarse reduction
                  N := N - 10;
                  X_Norm := X_Norm * 1024.0;      -- exact on binary machine
               end loop Coarse2;
            Fine2:
               while X_Norm < 0.5 loop          -- fine reduction
                  N := N - 1;
                  X_Norm := X_Norm * 2.0;      -- exact on binary machine
               end loop Fine2;
         end if;
         Exponent := N;
         MANTISSA := X_Norm;
      end Defloat;

   begin
      Defloat( Input, Sign, F, N );
      Znum := F - 0.5;
      if F > C0 then
         Znum := Znum - 0.5;
         Zden := F * 0.5 + 0.5;
      else
         N := N - 1;
         Zden := Znum * 0.5 + 0.5;
      end if;
      Z := Znum / Zden;
      if N = 0 then
         Inter_Result := R( Z );
      else
         Xn := Inputs(N);
         Inter_Result := (Xn * C2 + R( Z )) + Xn * C1;
      end if;
      Result := Outputs(Inter_Result);
      return Result;
   end Nat_Log;

end Cody_Natural_Log;
```

```
separate (Polynomials.Cody_Waite)
package body Cody_Log_Base_N is

   package Local_Natural_Log is new Cody_Natural_Log( Inputs  => Inputs,
                                                      Outputs => Outputs);

   package body Log_Base_N is

      One_Over_Base_Log : constant Outputs := 1.0 /
                          Local_Natural_Log.Nat_Log( Inputs(Base_N) );

      function Log_N ( Input : Inputs ) return Outputs is
      begin
         return Local_Natural_Log.Nat_Log( Input ) * One_Over_Base_Log;
      end Log_N;

   end Log_Base_N;

end Cody_Log_Base_N;
```

```
separate (Polynomials)
package body Reduction_Operations is

   function Sine_Reduction( Input : Inputs ) return Inputs is
      Result           : Inputs;
   begin
      if Input > Quarter_Cycle then
         Result := Half_Cycle - Input;
      elsif Input < - Quarter_Cycle then
         Result := - Half_Cycle - Input;
      else
         Result := Input;
      end if;
      return Result;
   end Sine_Reduction;

   function Cosine_Reduction( Input : Inputs ) return Inputs is
      Result : Inputs;
   begin
      return abs( Input );
   end Cosine_Reduction;

end Reduction_Operations;
```

(This page left intentionally blank.)

3.3.6.9 QUATERNION_OPERATIONS (PACKAGE BODY) TLCSC (CATALOG #P127-0)

This part, which is designed as an Ada package, contains bodies for for all CAMP parts which can be used on quaternions. A quaternion represents the orientation of frame xyz to frame XYZ. This part applies to missile navigation.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.9.1 REQUIREMENTS ALLOCATION

N/A

3.3.6.9.2 LOCAL ENTITIES DESIGN

None.

3.3.6.9.3 INPUT/OUTPUT

None.

3.3.6.9.4 LOCAL DATA

None.

3.3.6.9.5 PROCESS CONTROL

Not applicable.

3.3.6.9.6 PROCESSING

The following describes the processing performed by this part:

```
package body Quaternion_Operations is

   function Quaternion_Computed_From_Euler_Angles
                        (Euler_Angles  : Euler_Angle_Vectors)
                        return Quaternion_Vectors is separate;

   function Normalized_Quaternion (Quaternion : Quaternion_Vectors)
                                  return Quaternion_Vectors is separate;

end Quaternion_Operations;
```

3.3.6.9.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.9.8 LIMITATIONS

None.

3.3.6.9.9 LLCSC DESIGN

None.

3.3.6.9.10 UNIT DESIGN

3.3.6.9.10.1 QUATERNION_COMPUTED_FROM_EULER_ANGLES (FUNCTION BODY) UNIT DESIGN (CATALOG #P129-0)

This part computes the unit quaternion, Q, that represents the orientation of frame xyz with respect to XYZ (i.e. Q rotates XYZ into xyz) given the Euler angles relating xyz to XYZ.

3.3.6.9.10.1.1 REQUIREMENTS ALLOCATION

N/A

3.3.6.9.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.9.10.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part (as defined in the specification header):

Name	Type	Description
Euler_Angle_Indices	discrete type	Data type representing the index to the vector Euler_Angle_Vectors which has values such as Psi, Theta, and Phi.
Angles	floating point type	Data type for the elements of the Euler angle vector.
Euler_Angle_Vectors	array	Data type representing the Euler angles.

Data objects:

The following table summarizes the generic formal objects required by this part (as defined in the specification header):

Name	Type	Value	Description
Psi	Euler_Angle_Indices	'FIRST	An index that indexes "Euler_Angles" to extract the first Euler angle rotation that rotates XYZ into X'Y'Z' by rotating XYZ thru the angle psi about the Z axis.
Theta	Euler_Angle_Indices	'SUCC(psi)	An index that indexes "Euler_Angles" to extract the second Euler angle rotation that rotates X'Y'Z' into X''Y''Z'' by rotating X'Y'Z' thru the angle theta about the Y' axis.
Phi	Euler_Angle_Indices	'LAST	An index that indexes "Euler_Angles" to extract the third Euler angle rotation that rotates X''Y''Z'' into xyz by rotating X''Y''Z'' thru the angle phi about the X'' axis.

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
Sin_Cos	procedure	Procedure returning the sine and cosine of an euler angle (of type "Angles")

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Euler_Angles	Euler_Angle_Vectors	In	This value is a vector representing the euler angles.

3.3.6.9.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Quaternion	Quaternion_Vectors	N.A.	This 1X4 array contains the quaternion that will be computed and returned.
Cos_Psi_Div_2	Sin_Cos_Ratio	N.A.	An object for holding the value cosine(psi)/2.
Cos_Theta_Div_2	Sin_Cos_Ratio	N.A.	An object for holding the value cosine(theta)/2.
Cos_Phi_Div_2	Sin_Cos_Ratio	N.A.	An object for holding the value cosine(phi)/2.
Sin_Psi_Div_2	Sin_Cos_Ratio	N.A.	An object for holding the value sin(psi)/2.
Sin_Theta_Div_2	Sin_Cos_Ratio	N.A.	An object for holding the value sin(theta)/2.
Sin_Phi_Div_2	Sin_Cos_Ratio	N.A.	An object for holding the value sin(phi)/2.

3.3.6.9.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.9.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Quaternion_Operations)

function Quaternion_Computed_From_Euler_Angles
                      (Euler_Angles  : Euler_Angle_Vectors)
                      return Quaternion_Vectors is


   Quaternion       : Quaternion_Vectors;

   Cos_Psi_Div_2    : Sin_Cos_Ratio;
   Cos_Theta_Div_2  : Sin_Cos_Ratio;
   Cos_Phi_Div_2    : Sin_Cos_Ratio;
   Sin_Psi_Div_2    : Sin_Cos_Ratio;
   Sin_Theta_Div_2  : Sin_Cos_Ratio;
   Sin_Phi_Div_2    : Sin_Cos_Ratio;


begin

   Sin_Cos( Euler_Angles(Psi) * 0.5,   Sin_Psi_Div_2,   Cos_Psi_Div_2 );
   Sin_Cos( Euler_Angles(Theta) * 0.5, Sin_Theta_Div_2, Cos_Theta_Div_2 );
   Sin_Cos( Euler_Angles(Phi) * 0.5,   Sin_Phi_Div_2,   Cos_Phi_Div_2 );

   Quaternion(Q0) :=   Cos_Psi_Div_2 * Cos_Theta_Div_2 * Cos_Phi_Div_2
                     + Sin_Psi_Div_2 * Sin_Theta_Div_2 * Sin_Phi_Div_2;

   Quaternion(Q1) :=   Cos_Psi_Div_2 * Cos_Theta_Div_2 * Sin_Phi_Div_2
                     - Sin_Psi_Div_2 * Sin_Theta_Div_2 * Cos_Phi_Div_2;

   Quaternion(Q2) :=   Cos_Psi_Div_2 * Sin_Theta_Div_2 * Cos_Phi_Div_2
                     + Sin_Psi_Div_2 * Cos_Theta_Div_2 * Sin_Phi_Div_2;

   Quaternion(Q3) :=   Sin_Psi_Div_2 * Cos_Theta_Div_2 * Cos_Phi_Div_2
                     - Cos_Psi_Div_2 * Sin_Theta_Div_2 * Sin_Phi_Div_2;

   return Quaternion;
end Quaternion_Computed_From_Euler_Angles;
```

3.3.6.9.10.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.9.10.1.8 LIMITATIONS

None.

3.3.6.9.10.2 NORMALIZED_QUATERNION (FUNCTION BODY) UNIT DESIGN (CATALOG #P130-0)

This function normalizes a Quaternion when applied repeatedly. One iteration will not (in most cases) normalize the Quaternion. The frequency of execution is dependent upon the desired accuracy, the length of the time interval between updates, and other application-dependent factors. This part is usually applied repeatedly over time.

3.3.6.9.10.2.1 REQUIREMENTS ALLOCATION

N/A

3.3.6.9.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.9.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Quaternion	Quaternion_Vectors	In	This value is a vector representing a quaternion vector.

3.3.6.9.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Factor	Real	N.A.	This object is used to store a temporary value.
Answer	Quaternion_Vectors	N.A.	This object is the quaternion vector that is computed and returned.

3.3.6.9.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.9.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Quaternion_Operations)

function Normalized_Quaternion (Quaternion : Quaternion_Vectors)
                                return Quaternion_Vectors is

   Factor    : Real;
   Answer    : Quaternion_Vectors;


begin

   Factor := Real( 1.5 - (  (Quaternion(Q0) * Quaternion(Q0))
                           +(Quaternion(Q1) * Quaternion(Q1))
                           +(Quaternion(Q2) * Quaternion(Q2))
                           +(Quaternion(Q3) * Quaternion(Q3))  )
                          * Sin_Cos_Ratio(0.5));


   Answer(Q0) := Quaternion(Q0) * Factor;
   Answer(Q1) := Quaternion(Q1) * Factor;
   Answer(Q2) := Quaternion(Q2) * Factor;
   Answer(Q3) := Quaternion(Q3) * Factor;

   return Answer;

end Normalized_Quaternion;
```

3.3.6.9.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

Subprograms and task entries:

The following table summarizes the subprograms and task entries required by this part and defined elsewhere in the parent top level component:

Name	Type	Description
"*"	function	Function multiplying a type Sin_Cos_Ratio by a type Real returning type Sin_Cos_Ratio.

Data types:

The following table summarizes the types required by this part and defined elsewhere in the parent top level component:

Name	Type	Description
Quaternion_Indices	discrete type	Data type representing element indexes for the quaternion vector.
Real	floating point type	Data type used to compute a temporary value. This value is actually a Sin_Cos_Ratio; however, rounding errors can cause a constraint error. The final quaternions that are computed will not produce a constraint error if the |value| > 1.
Sin_Cos_Ratio	floating point type	Data type of elements of the quaternion vector.
Quaternion_Vectors	array	Data type representing the quaternions.

Data objects:

The following table summarizes the objects required by this part and defined elsewhere in the parent top level component:

Name	Type	Value	Description
Q0	Quaternion_Indices	'FIRST	An index that indexes "Quaternion" to extract the first quaternion element, which is the scalar part of a quaternion.
Q1	Quaternion_Indices	'SUCC(q0)	An index that indexes "Quaternion" to extract the second quaternion element, which is the first component of the vector.
Q2	Quaternion_Indices	'SUCC(q1)	An index that indexes "Quaternion" to extract the third quaternion element, which is the second component of the vector.
Q3	Quaternion_Indices	'LAST	An index that indexes "Quaternion" to extract the fourth quaternion element, which is the third component of the vector.

3.3.6.9.10.2.8 LIMITATIONS

None.

3.3.6.9.10.3 "*" (FUNCTION BODY) UNIT DESIGN (CATALOG #P126-0)

This generic function computes the product of two quaternions.

3.3.6.9.10.3.1 REQUIREMENTS ALLOCATION

N/A

3.3.6.9.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.9.10.3.3 INPUT/OUTPUT

GENERIC PARAMETERS:

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Quaternion_A	Quaternion_Vectors	In	This value is a vector representing a quaternion vector.
Quaternion_B	Quaternion_Vectors	In	This value is a vector representing a quaternion vector.

3.3.6.9.10.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Quat_C	Quaternion_Vectors	N.A.	The quaternion that is computed.

3.3.6.9.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.9.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
function "*" (Quaternion_A : Quaternion_Vectors;
              Quaternion_B : Quaternion_Vectors)
             return Quaternion_Vectors is

   Quat_C    : Quaternion_Vectors;

begin

   Quat_C(Q0) :=   Quaternion_A(Q0) * Quaternion_B(Q0)
                 - Quaternion_A(Q1) * Quaternion_B(Q1)
                 - Quaternion_A(Q2) * Quaternion_B(Q2)
                 - Quaternion_A(Q3) * Quaternion_B(Q3);

   Quat_C(Q1) :=   Quaternion_A(Q2) * Quaternion_B(Q3)
                 - Quaternion_A(Q3) * Quaternion_B(Q2)
                 + Quaternion_A(Q0) * Quaternion_B(Q1)
                 + Quaternion_A(Q1) * Quaternion_B(Q0);

   Quat_C(Q2) :=   Quaternion_A(Q3) * Quaternion_B(Q1)
                 - Quaternion_A(Q1) * Quaternion_B(Q3)
                 + Quaternion_A(Q2) * Quaternion_B(Q0)
                 + Quaternion_A(Q0) * Quaternion_B(Q2);

   Quat_C(Q3) :=   Quaternion_A(Q1) * Quaternion_B(Q2)
                 - Quaternion_A(Q2) * Quaternion_B(Q1)
                 + Quaternion_A(Q3) * Quaternion_B(Q0)
                 + Quaternion_A(Q0) * Quaternion_B(Q3);


   return Quat_C;


end "*";
```

3.3.6.9.10.3.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.9.10.3.8 LIMITATIONS

None.

```
package body Quaternion_Operations is


   function Quaternion_Computed_From_Euler_Angles
                        (Euler_Angles  : Euler_Angle_Vectors)
                        return Quaternion_Vectors is separate;

   function Normalized_Quaternion (Quaternion : Quaternion_Vectors)
                                   return Quaternion_Vectors is separate;

pragma PAGE;
function "*" (Quaternion_A : Quaternion_Vectors;
              Quaternion_B : Quaternion_Vectors)
             return Quaternion_Vectors is

   Quat_C    : Quaternion_Vectors;

begin

   Quat_C(Q0) :=   Quaternion_A(Q0) * Quaternion_B(Q0)
                 - Quaternion_A(Q1) * Quaternion_B(Q1)
                 - Quaternion_A(Q2) * Quaternion_B(Q2)
                 - Quaternion_A(Q3) * Quaternion_B(Q3);

   Quat_C(Q1) :=   Quaternion_A(Q2) * Quaternion_B(Q3)
                 - Quaternion_A(Q3) * Quaternion_B(Q2)
                 + Quaternion_A(Q0) * Quaternion_B(Q1)
                 + Quaternion_A(Q1) * Quaternion_B(Q0);

   Quat_C(Q2) :=   Quaternion_A(Q3) * Quaternion_B(Q1)
                 - Quaternion_A(Q1) * Quaternion_B(Q3)
                 + Quaternion_A(Q2) * Quaternion_B(Q0)
                 + Quaternion_A(Q0) * Quaternion_B(Q2);

   Quat_C(Q3) :=   Quaternion_A(Q1) * Quaternion_B(Q2)
                 - Quaternion_A(Q2) * Quaternion_B(Q1)
                 + Quaternion_A(Q3) * Quaternion_B(Q0)
                 + Quaternion_A(Q0) * Quaternion_B(Q3);


   return Quat_C;


end "*";

end Quaternion_Operations;
```

```
separate (Quaternion_Operations)

function Quaternion_Computed_From_Euler_Angles
                     (Euler_Angles  : Euler_Angle_Vectors)
                     return Quaternion_Vectors is


   Quaternion       : Quaternion_Vectors;

   Cos_Psi_Div_2    : Sin_Cos_Ratio;
   Cos_Theta_Div_2  : Sin_Cos_Ratio;
   Cos_Phi_Div_2    : Sin_Cos_Ratio;
   Sin_Psi_Div_2    : Sin_Cos_Ratio;
   Sin_Theta_Div_2  : Sin_Cos_Ratio;
   Sin_Phi_Div_2    : Sin_Cos_Ratio;


begin

   Sin_Cos( Euler_Angles(Psi) * 0.5,   Sin_Psi_Div_2,   Cos_Psi_Div_2 );
   Sin_Cos( Euler_Angles(Theta) * 0.5, Sin_Theta_Div_2, Cos_Theta_Div_2 );
   Sin_Cos( Euler_Angles(Phi) * 0.5,   Sin_Phi_Div_2,   Cos_Phi_Div_2 );

   Quaternion(Q0) :=   Cos_Psi_Div_2 * Cos_Theta_Div_2 * Cos_Phi_Div_2
                     + Sin_Psi_Div_2 * Sin_Theta_Div_2 * Sin_Phi_Div_2;

   Quaternion(Q1) :=   Cos_Psi_Div_2 * Cos_Theta_Div_2 * Sin_Phi_Div_2
                     - Sin_Psi_Div_2 * Sin_Theta_Div_2 * Cos_Phi_Div_2;

   Quaternion(Q2) :=   Cos_Psi_Div_2 * Sin_Theta_Div_2 * Cos_Phi_Div_2
                     + Sin_Psi_Div_2 * Cos_Theta_Div_2 * Sin_Phi_Div_2;

   Quaternion(Q3) :=   Sin_Psi_Div_2 * Cos_Theta_Div_2 * Cos_Phi_Div_2
                     - Cos_Psi_Div_2 * Sin_Theta_Div_2 * Sin_Phi_Div_2;

   return Quaternion;
end Quaternion_Computed_From_Euler_Angles;
```

```
separate (Quaternion_Operations)

function Normaiized_Quaternion (Quaternion : Quaternion_Vectors)
                                return Quaternion_Vectors is

   Factor    : Real;
   Answer    : Quaternion_Vectors;


begin

   Factor := Real( 1.5 - (  (Quaternion(Q0) * Quaternion(Q0))
                           +(Quaternion(Q1) * Quaternion(Q1))
                           +(Quaternion(Q2) * Quaternion(Q2))
                           +(Quaternion(Q3) * Quaternion(Q3))  )
                          * Sin_Cos_Ratio(0.5));


   Answer(Q0) := Quaternion(Q0) * Factor;
   Answer(Q1) := Quaternion(Q1) * Factor;
   Answer(Q2) := Quaternion(Q2) * Factor;
   Answer(Q3) := Quaternion(Q3) * Factor;

   return Answer;

end Normalized_Quaternion;
```

(This page left intentionally blank.)

SUPPLEMENTARY

INFORMATION

DEPARTMENT OF THE AIR FORCE
WRIGHT LABORATORY (AFSC)
EGLIN AIR FORCE BASE, FLORIDA, 32542-5434

ERRATA
AD-B120258

REPLY TO ATTN OF: MNOI

13 Feb 92

SUBJECT: Removal of Distribution Statement and Export-Control Warning Notices

TO: Defense Technical Information Center
ATTN: DTIC/HAR (Mr William Bush)
Bldg 5, Cameron Station
Alexandria, VA 22304-6145

1. The following technical reports have been approved for public release by the local Public Affairs Office (copy attached).

	Technical Report Number	AD Number
1.	88-18-Vol-4	ADB 120 251
2.	88-18-Vol-5	ADB 120 252
3.	88-18-Vol-6	ADB 120 253
4.	88-25-Vol-1	ADB 120 309
5.	88-25-Vol-2	ADB 120 310
6.	88-62-Vol-1	ADB 129 568
7.	88-62-Vol-2	ADB 129 569
8.	88-62-Vol-3	ADB 129-570
9.	85-93-Vol-1	ADB 102-654
10.	85-93-Vol-2	ADB 102-655
11.	85-93-Vol-3	ADB 102-656
12.	88-18-Vol-1	ADB 120 248
13.	88-18-Vol-2	ADB 120 249
14.	88-18-Vol-7	ADB 120 254
15.	88-18-Vol-8	ADB 120 255
16.	88-18-Vol-9	ADB 120 256
17.	88-18-Vol-10	ADB 120 257
18.	88-18-Vol-11	ADB 120 258
19.	88-18-Vol-12	ADB 120 259

2. If you have any questions regarding this request call me at DSN 872-4620.

LYNN S. WARGO
Chief, Scientific and Technical
Information Branch

1 Atch
AFDIC/PA Ltr, dtd 30 Jan 92

ERRATA

DEPARTMENT OF THE AIR FORCE
HEADQUARTERS AIR FORCE DEVELOPMENT TEST CENTER (AFSC)
EGLIN AIR FORCE BASE, FLORIDA 32542-5000

REPLY TO ATTN OF: PA (Jim Swinson, 882-3931) 30 January 1992

SUBJECT: Clearance for Public Release

TO: WL/MNA

The following technical reports have been reviewed and are approved for public release: AFATL-TR-88-18 (Volumes 1 & 2), AFATL-TR-88-18 (Volumes 4 thru 12), AFATL-TR-88-25 (Volumes 1 & 2), AFATL-TR-88-62 (Volumes 1 thru 3) and AFATL-TR-85-93 (Volumes 1 thru 3).

Virginia N. Pribyla

VIRGINIA N. PRIBYLA, Lt Col, USAF
Chief of Public Affairs

AFDTC/PA 92-039