APPENDIX P

Derived Parameter Specification

List of Figures

List of Tables

Acronyms

AT&T	American Telephone and Telegraph
BNF	Backus-Naur Form
Lex	Lexicon
TMATS	Telemetry Attributes Transfer Standard
Yacc	Yet Another Compiler Compiler

This page intentionally left blank.

APPENDIX P

Derived Parameter Specification

1.0 Derived Parameter Definition

Derived parameters are measurements that do not appear in any data stream; instead, they are calculated from telemetry measurements in a data stream, numeric constants, and/or other derived measurements. In a Telemetry Attributes Transfer Standard (TMATS) file, derived measurements will only have entries in the C group; the other TMATS groups containing measurement names that link to C group entries only include telemetry measurements.

Derived parameters are defined using the Algorithm Type (C-d\DPAT) and Algorithm (C-d\DPA) attributes in the Derived Parameter section of the TMATS C group. They can be defined in one of two methods. The first method to specify the name of an algorithm ("function style") and the second method is to specify a text string of the algorithm itself ("formula style"). Both of these methods are currently used in telemetry processing systems.

In function style, Algorithm Type is set to "N" and Algorithm contains the name of a function, which will be one of the mathematical functions or operators as defined in the derived algorithm grammar shown in this appendix. The Input Measurand attributes (C-d\DP\N and C-d\DP-n) and Input Constant attributes (C-d\DPC\N and C-d\DPC-n) are used to specify the arguments needed by the named function (measurements and numeric constants, respectively, as defined in the derived algorithm grammar in this appendix). The Trigger Measurand and Number of Occurrences attributes are used to specify when and how often the derived parameter will be calculated.

In formula style, Algorithm Type is set to "A" and Algorithm contains the actual function, given according to the derived algorithm grammar defined in this appendix. The Input Measurand attributes and Input Constant attributes are not used. The Trigger Measurand and Number of Occurrences attributes are used to specify when and how often the derived parameter will be calculated.

2.0 Derived Algorithm Grammar: Components

Derived algorithm grammar is from the four components listed below. The derived algorithm may be any combination of operators, functions, measurements, and numeric constants strung together using the guidelines in this document to create complex mathematical expressions (see Subparagraph 6.2). Sample syntaxes for the Yet Another Compiler Compiler (Yacc) grammar and Lexicon (Lex) grammar are provided in Section 8.0.

a. Operators (Section 3.0)

b. Numeric Constants (Section 4.0)

c. Measurements (Section 5.0)

d. Mathematical Functions (Section 6.0).

3.0 Operators

Operators are simply mathematical functions that have a special syntax in the grammar. They have operator symbol(s) that have well-defined arguments and return a value as a result. Logical operators are merely functions that return a value of 0 and non-zero for false and true respectively.

3.1 Arithmetic Operators

Table P-1. Arithmetic Operators

Operator	Description	Example
+	Addition (Sum)	A + B
-	Subtraction (Difference)	A – B
*	Multiplication (Product)	A * B
/	Division (Quotient)	A / B
%	Modulus (Remainder)	A % B
**	Exponentiation	A ** B

3.2 Bit Manipulation Operators

Table P-2. Bit Manipulation Operators

Operator	Description	Example
|	Bit-wise OR	A | B
&	Bit-wise AND	A & B
^	Bit-wise XOR	A ^ B
~	Bit-wise NOT	~A
<<	Bit-wise Left Shift	A << B
>>	Bit-wise Right Shift	A >> B

3.3 Relational Operators

Table P-3. Relational Operators

Operator	Description	Example
= =	Equal To	A = = B
!=	Not Equal To	A != B
<=	Less Than or Equal To	A <= B
>=	Greater Than or Equal To	A >= B
<	Less Than	A < B
>	Greater Than	A > B
||	Logical OR	A || B
&&	Logical AND	A && B
!	Logical NOT (Negation)	!A

3.4 Ternary (if then else) Operator

Table P-4. Ternary (if then else) Operator

Operator	Description	Example
?:	Ternary Operator (if-then-else)	A ? B : C

3.5 Associativity Operator

Table P-5. Associativity Operator

Operator	Description	Example
( )	Associativity	(A + B) * C

3.6 Precedence and Associativity of Operators From Highest to Lowest

Table P-6. Precedence and Associativity of Operators from Highest to Lowest

Operators	Associativity
( )	Left to right
-(UNARY)	Right to left
! ~	Right to left
**	Left to right
&	Left to right
^	Left to right
|	Left to right
* / %	Left to right
+ -	Left to right
<< >>	Left to right
< > <= >=	Left to right
= = !=	Left to right
&&	Left to right
||	Left to right
?:	Right to left
,	Left to right

4.0 Numeric Constants

Numeric constants are simply numbers used in the calculations.

Table P-7 Numeric Constants (Examples)

Description	Examples
Any string of characters that contains only numerals	1234 0
Any string of characters that contains only numerals and a-f preceded by "0x" (hex)	0x12ab 0x1

Any string of characters that contains only numerals and a single ".".	1.2 1. .2
Any string of characters that contains only numerals, in scientific notation.	1.0E+10 10E-10 .1e6
Note: As in the TMATS standard itself, alphanumeric data items are case insensitive; either upper or lower case characters are allowed.	

5.0 Measurements

Measurements may be telemetry measurements or other derived measurements.

Table P-8. Measurements (Examples)

Description	Examples
Any string of characters beginning with an alphabetic character and containing only alphanumerics and "$_"	A00.1 A$1
Any string of characters that is quoted with " and does not contain ".	"0001" “measurement ’quoted’, though this is insane - it is legal”
Any string of characters quoted with ' and does not contain '.	'Air Speed'
Any string of characters that contains only numerals and at least one alphabetic character. This differs from hex because it does not begin with “0x”.	00A1 0X (this is ok, because it does not have a number after “0X”)
Note: As in the TMATS standard itself, alphanumeric data items are case insensitive; either upper or lower case characters are allowed.	

6.0 Mathematical Functions

6.1 Mathematical Function Format

Mathematical functions are numerical functions that take some input, perform a specific calculation, and return a value as the result. Each mathematical function has the form “name(arg1,arg2,...)” that identifies a well-defined name and contains argument(s) that are separated by commas and surrounded by parentheses. A list of selected mathematical functions is provided in Table P-9.

6.2 Complex Use of Functions

Examples of how functions can be used in mathematical expressions are:

a. A*(SIN(B/C)+D)
b. A*3.0
c. "0001"*A+~B
d. A<B || B<<C ? D : E

Table P-9. Table of Selected Mathematical Functions	
Name	**Description**
acos(x)	cos-1(x) in range [0,π], x Є [-1,1].
asin(x)	sin-1(x) in range [-π/2, π/2], x Є [-1,1].
atan(x)	tan-1(x) in range [-π/2, π/2]
atan2(y,x)	tan-1(y/x) in range [-π, π]
ceil(x)	smallest integer not less than x
cos(x)	cosine of x
cosh(x)	hyperbolic cosine of x
exp(x)	exponential function, computes ex
fabs(x)	absolute value |x|
floor(x)	largest integer not greater than x
fmod(x)	floating point remainder
frexpx(d)	Find x in [.5,1] and y so that d = x*pow(2,y), return x
frexpy(d)	Find x in [.5,1] and y so that d = x*pow(2,y), return y
ldexp(d,i)	returns d*pow(2,i)
log(x)	natural logarithm ln(x), x > 0
log10(x)	base-10 logarithm log10(x), x > 0
max(x,y)	if x>y, then return x, else return y
min(x,y)	if x<y, then return x, else return y
modfd(d)	returns integral part of d
modfp(d)	returns fractional part of d
pow(x,y)	compute a value taken to an exponent, xy. An error occurs when x<=0 and y <= 0 or x < 0 and y is not an integer
sin(x)	sine of x
sinh(x)	hyperbolic sine of x
sqrt(x)	square root √x, x>= 0
tan(x)	tangent of x
tanh(x)	hyperbolic tangent of x

7.0 Derived Grammar Syntax Overview

The following grammar, strictly speaking, does not match the C language. Although loosely based on C, the grammar attempts to follow the "spirit" of the C language. The grammar contains three terminal symbols (MEASUREMENT, NUMERIC_CONSTANT, and FUNCTION_NAME) not defined here, but easily understood by their names. The grammar contains two non-terminals, expression and expression-list, which define the entire grammar. The "|" operator used in the grammar denotes a choice meaning "this or that or …" Quoted strings are literal tokens of the grammar.

```
expression:
        expression '+' expression
        | expression '-' expression
        | expression '*' expression
        | expression '/'  expression
        | expression '|' expression
        | expression '&' expression
        | expression '%' expression
        | expression '**' expression
        | expression '?' expression ':' expression
        | expression '<' expression
        | expression '>' expression
        | expression '<=' expression
        | expression '>=' expression
        | expression '!=' expression
        | expression '==' expression
        | expression '&&' expression
        | expression '||' expression
        | '-' expression
        | '!' expression
        | '~' expression
        | ' (' expression ') '
        | MEASUREMENT
        | NUMERIC_CONSTANT
        | FUNCTION_NAME '(' expression_list ')'
        | FUNCTION_NAME ' (' ')'

        expression-list:
        expression
        | expression-list ',' expression
```

Figure P-1. Grammar Syntax

8.0 Grammar Examples

Examples of Yacc and Lex grammar are shown in Figure P-2 and Figure P-3, respectively. The grammar will recognize the derived syntax; that is, they will report whether or not a given text string is valid syntax; however, the examples are not intended to be complete; in other words, they will not compile or perform the calculation. The user needs only to build a program around them in order to use them; a simple example "main" is shown in Figure P-4.

The Yacc is a parser generator developed by Stephen C. Johnson at American Telephone and Telegraph (AT&T) for the Unix operating system. It generates a parser, in C language code, based on an analytic grammar written in a notation similar to Backus-Naur Form (BNF). The Lex, a program that generates lexical analyzers, is commonly used along with the Yacc parser generator. Originally written by Eric Schmidt and Mike Lesk, Lex is the standard lexical

analyzer generator on many Unix systems. A tool exhibiting its behavior is specified as part of the Portable Operating System Interface standard.

```
%{

%}

%token ERR
%token NAME
%token CONSTANT

// Operator Precedence Rules (Lowest First, Highest Last)

%left ','
%right COND '?'
%left OR
%left AND
%left EQUAL NOTEQUAL
%left '<' '>' LESSEQUAL GREATEREQUAL
%left LSHIFT RSHIFT
%left '-' '+'
%left '*' '/' '%'
%left '|'
%left '^'
%left '&'
%left POWER
%right '!' '~'
%right UMINUS

// Definition of Rules

%%
expression:
        expression '+' expression
        | expression '-' expression
        | expression '*' expression
        | expression '^' expression
        | expression '&' expression
        | expression '%' expression
        | expression LSHIFT expression
        | expression RSHIFT expression
        | expression POWER expression
        | expression '?' expression ':' expression %prec COND
```

Figure P-2. Yacc Grammar Example, Page 1 of 2

```
        | '-' expression %prec UMINUS
        | '!' expression
        | '~' expression
        | '(' expression ')'
        | NAME
        | CONSTANT
        | NAME '(' expression_list ')'
        | NAME '(' ')'
        | expression '<' expression
        | expression '>' expression
        | expression LESSEQUAL expression
        | expression GREATEREQUAL expression
        | expression NOTEQUAL expression
        | expression EQUAL expression
        | expression OR expression
        | expression AND expression
        ;

expression_list:
        expression
        | expression_list ',' expression
        ;


%%
```

Figure P-3. Yacc Grammar Example, Page 2 of 2

```
%{
#include “y.tab.h”
%}

%%

[\t\n ]                 {}

\=\=                    { return(EQUAL); }              // Equal To
\!\=                    { return(NOTEQUAL); }                   // Not Equal To
\<\=                    { return(LESSEQUAL); }                  // Less Than or Equal To
\>\=                    { return(GREATEREQUAL); }       // Greater Than or Equal To
(\*\*)                  { return(POWER); }              // Power (FORTRANish)
\|\|                    { return(OR); }                         // Logical OR
\&\&                    { return(AND); }                // Logical AND
\<\<                    { return(LSHIFT); }             // Bitwise Left Shift
\>\>                    { return(RSHIFT); }             // Bitwise Right Shift

\>                      |                               // Greater Than
\<                      |                               // Less Than
\!                      |                               // Logical Negation
\?                      |                               // Ternary Operator ?
\:                      |                               // Ternary Operator :
\%                      |                               // Modulus (Remainder)
\,                      |                               // Comma Operator (function)
\*                      |                               // Multiplication (Product)
\/                      |                               // Division (Quotient)
\+                      |                               // Addition (Sum)
\-                      |                               // Subtraction (Difference)
\|                      |                               // Bitwise OR
\&                      |                               // Bitwise AND
\^                      |                               // Bitwise XOR
\~                      |                               // Bitwise NOT
\(                      |
\)                      { return(yytext[0]); }
```

Figure P-4. Lex Grammar Example, Page 1 of 2

```
([0][xX][0-9a-fA-F]+)|([0-9]+)                          {
                                        return(CONSTANT);
                }

(([0-9]+\.?[0-9]*)|([0-9]*\.[0-9]+))([eE][-+]?[0-9]+)? {
                                        return(CONSTANT);
                }

\"[^\"\n]*\"            |
\'[^\n]*\'              {
                                        return(NAME);
                        }
([0-9]+[a-zA-Z])?[a-zA-Z0-9$_\.]+   {
                                        return(NAME);
                        }


.                       { return(ERR); }            // Catchall Error

%%
```

Figure P-5. Lex Grammar, Page 2 of 2

```
yywrap()
{
  return 1;
}
yyerror(char *s)
{
  printf("error: %s\n",s);
}
main()
{
  yyparse();
}
```

Figure P-6. Example Program (Main)

9.0 Telemetry Attributes Transfer Standard (TMATS) Examples

In the following examples, input measurement names are in the form of MA, MB, and MC. Derived parameter names are in the form of DMA, DMB, and DMC.

9.1 TMATS Example 1

DMA = MA + MB

Function style

C-1\DCN:DMA;	Derived parameter
C-1\DCT:DER;	Derived conversion type
C-1\DPAT:N;	Name of algorithm will be given
C-1\DPA:+;	Addition operator
C-1\DPTM:MB;	Measurement MB triggers the calculation
C-1\DPNO:1;	Every sample of MB triggers the calculation
C-1\DP\N:2;	Two input measurements
C-1\DP-1:MA;	
C-1\DP-2:MB;	

Formula style

C-2\DCN:DMA;	
C-2\DCT:DER;	
C-2\DPAT:A;	Algorithm will be given
C-2\DPA:MA + MB;	Algorithm syntax
C-2\DPTM:MB;	
C-2\DPNO:1;	

9.2 TMATS Example 2

DMB = MC / MD

Function style

C-3\DCN:DMB;	Derived parameter
C-3\DCT:DER;	Derived conversion type
C-3\DPAT:N;	Name of algorithm will be given
C-3\DPA:/;	Division operator
C-3\DPTM:MD;	Measurement MD triggers the calculation
C-3\DPNO:1;	Every sample of MD triggers the calculation
C-3\DP\N:2;	Two input measurements
C-3\DP-1:MC;	
C-3\DP-2:MD;	

Note: In function style, the algorithm determines the meaning of the input measurements. In this example, the division algorithm assigns the first input measurement as the dividend and the second input measurement as the divisor.

Formula style

C-4\DCN:DMB;
C-4\DCT:DER;
C-4\DPAT:A; Algorithm will be given
C-4\DPA:MC / MD; Algorithm syntax
C-4\DPTM:MD;
C-4\DPNO:1;

9.3 TMATS Example 3

DMC = square root of ME

Function style

C-5\DCN:DMC; Derived parameter
C-5\DCT:DER; Derived conversion type
C-5\DPAT:N; Name of algorithm will be given
C-5\DPA:SQRT; Square root function
C-5\DP\N:1; One input measurement
C-5\DP-1:ME;

Formula style

C-6\DCN :DMC;
C-6\DCT :DER;
C-6\DPAT:A; Algorithm will be given
C-6\DPA:SQRT(ME); Algorithm syntax

Note: The trigger measurand is not given; there is only one input, which must trigger the calculation.

9.4 TMATS Example 4

DMD = MF*(SIN(MG/MH)+MJ)

Function style

C-7\DCN:XA; Derived parameter
C-7\DCT:DER; Derived conversion type
C-7\DPAT:N; Name of algorithm will be given
C-7\DPA:/; Division operator
C-7\DP\N:2; Two input measurements
C-7\DP-1:MG;
C-7\DP-2:MH;
C-8\DCN:XB; Derived parameter
C-8\DCT:DER; Derived conversion type
C-8\DPAT:N; Name of algorithm will be given

```
C-8\DPA:SIN;            Sine function
C-8\DP\N:1;             One input measurement
C-8\DP-1:XA;
C-9\DCN:XC;             Derived parameter
C-9\DCT:DER;            Derived conversion type
C-9\DPAT:N;             Name of algorithm will be given
C-9\DPA:+;              Addition operator
C-9\DP\N:2;             Two input measurements
C-9\DP-1:XB;
C-9\DP-2:MJ;
C-10\DCN:DMD;           Derived parameter
C-10\DCT:DER;           Derived conversion type
C-10\DPAT:N;            Name of algorithm will be given
C-10\DPA:*;             Multiplication operator
C-10\DP\N:2;            Two input measurements
C-10\DP-1:MF;
C-10\DP-2:XC;
```

Note: In this example, several steps are needed, each generating an intermediate result (XA, XB, and XC), before the derived parameter is obtained. This method is shown only for illustrative purposes and is not recommended. If this function is needed, a custom algorithm should be written to implement it. Then the function style could be used, as follows:

```
C-11\DCN:DMD;           Derived parameter
C-11\DCT:DER;           Derived conversion type
C-11\DPAT:N;            Name of algorithm will be given
C-11\DPA:NEWALG;        Name of custom algorithm
C-11\DPTM:MJ;
C-11\DPNO:1;
C-11\DP\N:4;            Four input measurements
C-11\DP-1:MF;
C-11\DP-2:MG;
C-11\DP-3:MH;
C-11\DP-4:MJ;
```

Formula style

```
C-12\DCN:DMD;
C-12\DCT:DER;
C-12\DPAT:A;            Algorithm will be given
C-12\DPA:MF*(SIN(MG/MH)+MJ);
C-12\DPTM:MJ;
C-12\DPNO:1;
```

10.0 Glossary of Terms

Backus-Naur Form: A metasyntax used to express context-free grammar; that is, a formal way to describe formal languages. John Backus and Peter Naur developed a context free grammar to define the syntax of a programming language by using two sets of rules: i.e., lexical rules and syntactic rules

Compiler: A computer program (or set of programs) that transforms source code written in a computer language (the source language) into another computer language (the target language, often having a binary form known as object code).

Compiler Compiler (Compiler Generator): A tool that creates a parser, interpreter, or compiler from some form of formal description. The earliest and still most common form of compiler-compiler is a parser generator, whose input is a grammar (usually in BNF) of a programming language, and whose generated output is the source code of a parser.

Computer Programs: Also called software programs, or just programs, are instructions for a computer.

Grammar: A set of formation rules that describe which strings formed from the alphabet of a formal language are syntactically valid within the language.

Interpreter: Normally means a computer program that executes instructions written in a programming language.

Parser Generator: See Compiler Compiler.

Parsing: The process of analyzing a sequence of tokens (for example, words) to determine their grammatical structure with respect to a given (more or less) formal grammar.

Programming Language: A machine-readable artificial language designed to express computations that can be performed by a machine, particularly a computer.

Source Code: Any collection of statements or declarations written in some human-readable computer programming language.

Unix: A computer operating system originally developed in 1969 by a group of AT&T employees at Bell Labs.

Yet Another: In hacker jargon, the use of yet another as a way of padding out an acronym is fairly common. It was first used by Stephen C. Johnson in the late 1970s in naming Yacc as a humorous reference to the proliferation of such compiler-compilers at the time.

Yet Another Compiler Compiler (Yacc): Supplied with Unix and Unix-like systems.

****** END OF APPENDIX P ******