

Interface Specifications for the SCR (A-7E) Application Data Types Module

P. C. CLEMENTS, S. R. FAULK, AND D. L. PARNAS*

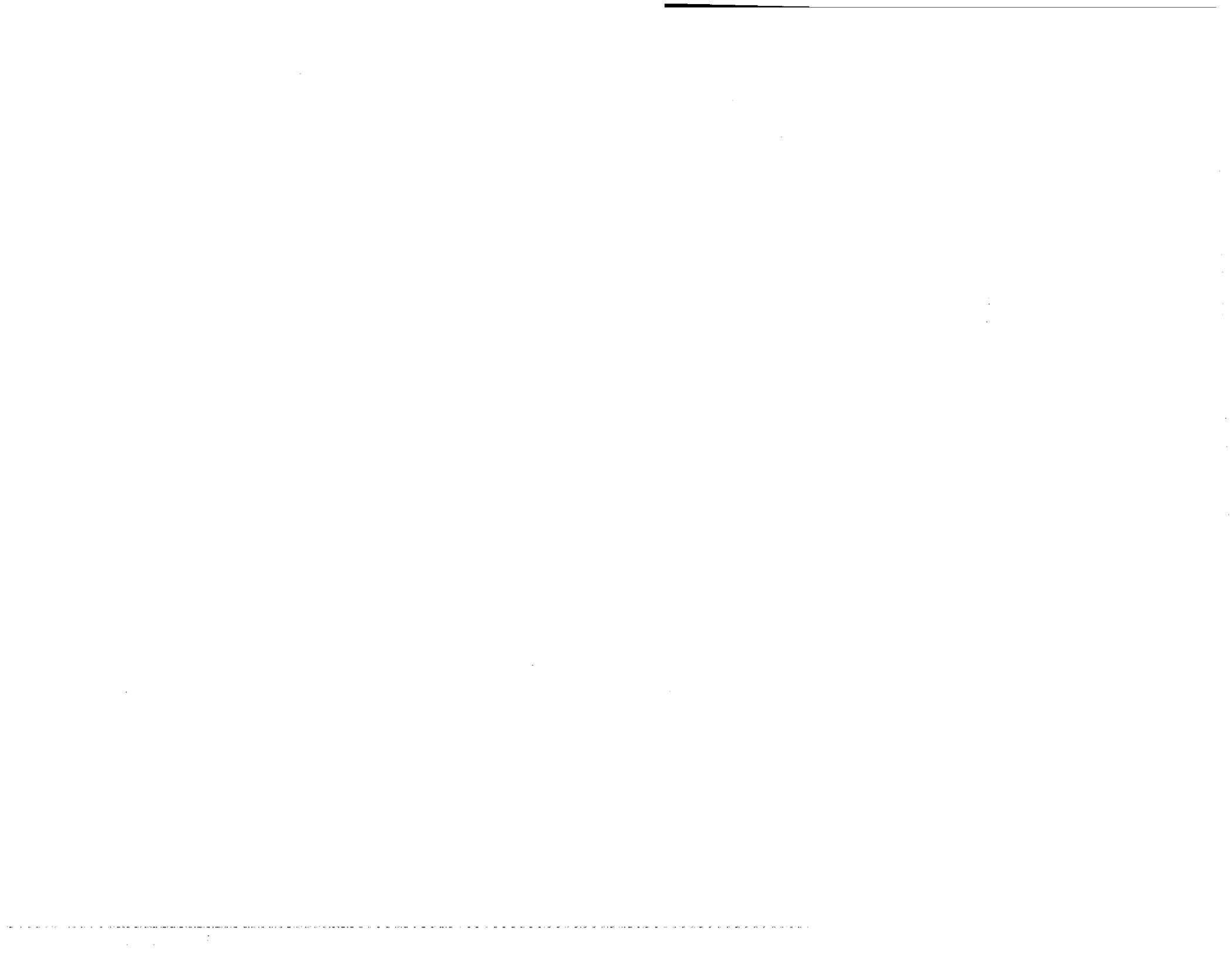
*Computer Science and Systems Branch
Information Technology Division*

**Also at University of Victoria
Victoria, B.C.*

August 23, 1983



NAVAL RESEARCH LABORATORY
Washington, D.C.



REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NRL Report 8734	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) INTERFACE SPECIFICATIONS FOR THE SCR (A-7E) APPLICATION DATA TYPES MODULE		5. TYPE OF REPORT & PERIOD COVERED Interim report on a continuing NRL problem
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) P.C. Clements, S.R. Faulk, and D.L. Parnas*		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Research Laboratory Washington, DC 20375		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62721N; XF21242101; 75-0106-03
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE August 23, 1983
		13. NUMBER OF PAGES 31
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES *Also at University of Victoria, Victoria, B.C.		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
Abstract data types	Modular decomposition	Software maintenance
Abstract interfaces	Modules	Software specifications
Avionics software	Real-time systems	
Information hiding	Software engineering	
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)		
<p>This report describes the programmer interface to a set of avionics-oriented abstract data types implemented in software. The Application Data Types module is part of NRL's Software Cost Reduction (SCR) project, to demonstrate the feasibility of applying advanced software engineering techniques to complex real-time systems to simplify maintenance. The Application Data Types module allows operations on data independent of the representation. In the case of</p> <p style="text-align: right;">(Continued)</p>		

20. ABSTRACT (Continued)

numeric abstract types, which represent physical quantities such as speed or distance, arithmetic operations may be performed independent of the units of physical measure. This allows the rest of the application software to remain unchanged even when representation decisions change about these data.

This report contains the abstract interface specifications for all of the facilities provided to users by this module. It serves as development and maintenance documentation for the SCR software design, and it is also intended as a model for other people interested in applying the abstract interface approach on other software projects.

CONTENTS

AT.INTRO—INTRODUCTION	1
AT.NUM—NUMERIC ABSTRACT DATA TYPES AND OPERATIONS	3
AT.STE—STATE TRANSITION EVENT TYPE CLASS	12
AT.INDEX—INDICES TO THE DOCUMENT	19
ACKNOWLEDGMENTS	22
REFERENCES	22
APPENDIX 1—Interface Design Issues	23
APPENDIX 2—Implementation Notes	26
APPENDIX 3—Basic Assumptions	27
APPENDIX 4—Unimplemented Application Data Type Facilities	29
APPENDIX 5—Data Representation (Version) Catalog	30

INTERFACE SPECIFICATIONS FOR THE SCR (A-7E) APPLICATION DATA TYPES MODULE

INTRODUCTION

This report specifies the abstract interfaces for a software module that forms part of the Operational Flight Program for the Navy's A-7E aircraft. As the demonstration vehicle for the Naval Research Laboratory's Software Cost Reduction (SCR) project, the program is being designed and implemented in accordance with modern software engineering principles such as information-hiding, cooperating sequential processes, and abstract data typing.

The basic computing environment for the application software is provided by the Extended Computer module, specified in reference [EC]. The Extended Computer interface provides all the facilities necessary for handling data, as well as input/output (i/o), sequence, and process control.

Overview

The application Data Types (ADT) module provides facilities for handling those data types which (a) are useful to the application at hand (in this case, an embedded real-time avionics program); and (b) can be implemented independently of the host computer, i.e., are not provided by the Extended Computer Module.

Although the type classes provided by the module are fixed, operations are available to specify specific types within a type class, and to create and manipulate data objects of each type. The module comprises two submodules:

(1) Numeric Type Classes. This submodule provides those numeric data type classes that are considered to be generally useful to avionics applications; these type classes are used to represent physical quantities: acceleration (both scalar and vector), angle, angular rate, density, displacement, distance, pressure, speed, and velocity. Facilities are provided that allow manipulation of such quantities without regard to the units of measurement.

(2) State Transition Event (STE) Type Class. This submodule allows programs to create and operate on data types described as finite state machines. The domain of an STE variable is a set of states. The changing of a variable's state is an event that can be signaled and awaited.

Reader Prerequisites

This report is a companion to Chapter EC.DATA of [EC], and it is assumed that the reader is familiar with that chapter, particularly the notion of type classes, specific types, and other terminology used therein. The ADT interfaces are modeled after that submodule of the Extended Computer. To present as concise a document as possible, we frequently refer to that chapter rather than duplicate information that appears there.

This report follows the standard organization presented in [SO], as modified in the Introduction to [EC], and readers should also be familiar with that.

Manuscript approved April 19, 1983.

Referring to the Extended Computer

When a reference is made to Chapter EC.DATA of [EC], the following rules apply:

- (1) If the reference is made to an EC.DATA access program, then the name of the ADT access program that is being specified, as well as the parameters, parameter meanings, parameter requirements, undesired events, undesired event dictionary definitions, and program effects will be the same as those given for the access program in EC.DATA, except as noted.
- (2) Any information about registers given in a referenced section of EC.DATA should be ignored. This module does not provide registers.

AT.NUM NUMERIC ABSTRACT DATA TYPES AND OPERATIONS

AT.NUM.1 Introduction

AT.NUM.1.1 Overview

The type classes provided by this submodule give a means to represent and operate on physical quantities (such as a speed or an angle) without stating or assuming particular units of measurement. Conversion programs are provided to convert the entity to its real equivalent, given the unit of measurement desired.

The notion of type classes and specific types is explained in EC.DATA.1.2. The types provided by this module, as well as the available units of measurement for each, are listed in Table AT.NUM.a.

Table AT.NUM.a — Types Classes and Associated Units

Type Class	Unit	Unit Meanings
accel	fpss	Feet per second per second
	p	Normal gravity acceleration
accel-vec	<i>a</i>	
angle	deg	Degrees
	cir	Circles
	rad	Radians
		Angles may also be converted to a sine/cosine representation
angrate	deghour	Degrees per hour
	radsec	Radians per second
density	slcuft	Slugs per cubic foot
displacement	<i>b</i>	
distance	ft	Feet
	nmi	Nautical miles
pressure	inhg	Inches of mercury
speed	fps	Feet per second
	fpm	Feet per minute
	knot	Nautical miles per hour
velocity	<i>c</i>	

^aaccel-vec is a three-dimensional vector that may be converted into components of type accel.

^bDisplacement is a three-dimensional vector that may be converted into components of type distance.

^cVelocity is a three-dimensional vector that may be converted into components of type speed.

Specific types in this module are characterized only by the range and resolution of all entities of that type. Specific types are created via declarations. Each variable and constant must belong to a specific type.

AT.NUM.1.2 Literals

A value for a scalar type provided by this module can be specified by using a real literal and one of the conversion programs that convert real values to numeric scalar abstract types. The syntax is:

< program-name, real-literal >

The syntax for real-literal is given in EC.DATA.1.3. The value thus specified is that which would be returned by the program were the real given as the input parameter.

A value for a vector type can be specified similarly, by naming a program that produces a vector from constituent scalars, which are specified as shown above. The syntax is:

< program-name, scalar-literal, scalar-literal, scalar-literal >

AT.NUM.2 Interface Overview

AT.NUM.2.1 Declaration of Specific Types

The form is that of the ++DCL_TYPE++ program described in EC.DATA.2.1, with

- p1 as described there;
- p2 a typeclass as described in AT.NUM.4;
- p3 an attribute as described in AT.NUM.4;
- p4 as described there; and
- p5 a version as described in AT.NUM.4.

AT.NUM.2.2 Data Declarations

AT.NUM.2.2.1 Declaration of Variables and Constants

The form is that of the ++DCL_ENTITY++ program described in EC.DATA.2.2.1, with

- p1 and p2 as described there;
- p3, if supplied, an attribute as described in AT.NUM.4; and
- p4 and p5 as described there.

AT.NUM.2.2.2 Declaration of Arrays

The form is that of the ++DCL_ARRAY++ program described in EC.DATA.2.2.2, with

- p1 and p2 as described there;
- p3, if supplied, an attribute as described in AT.NUM.4; and
- p4, p5, and p6 as described there.

AT.NUM.2.3 Access Speed Ranking of Data

The ADT module can implement a "not-slower-than" relation between any two variables, constants, or arrays. The form is that of the ++RANK_DATA++ program described in EC.DATA.2.3.

AT.NUM.2.4 Operand Descriptions

All information in EC.DATA.2.4 applies, except for information concerning registers.

AT.NUM.2.5 Transfer Operations

The form is that of the +SET+ program described in EC.DATA.2.5; p1 and p2 must both belong to the same type class.

AT.NUM.2.6 Numeric Operations

AT.NUM.2.6.1 Numeric Comparison Operations

All programs listed in EC.DATA.2.6.1 apply. Parameters p1, p2, and p4 (if given) must all belong to the same type class. For operands of vector type classes, the following program effects apply:

+GT+ p3 = (p1 - p2) is positive and NOT (p1 = p2)*

+GEQ+ p3 = (p1 = p2)* OR (p1 - p2) is positive

+LT+ p3 = (p1 - p2) is negative and NOT (p1 = p2)*

+LEQ+ p3 = (p1 = p2)* OR (p1 - p2) is negative

AT.NUM.2.6.2 Numeric Calculations

All programs in section EC.DATA.2.6.2 apply. The following parameter information applies:

Parameters

+ADD+ all operands belong to the same AT-provided type class
 +ABSV+
 +COMPLE+
 +SUB+

+MUL+ one of p1 or p2 real, the other two operands belong to the same AT-provided type class; or, p3 speed; one of p1 and p2 timeint, the other accel; or p3 distance; one of p1 and p2 timeint, the other speed; or p3 angle; one of p1 and p2 timeint, the other angrate; or p3 displacement; one of p1 and p2 timeint, the other velocity; or p3 velocity; one of p1 and p2 timeint, the other accel-vec.

+DIV+ p1 and p2 belong to same AT scalar type class and p3 real; or, p1 and p3 belong to same AT-provided type class and p2 real; or,
 p1 speed, p2 timeint, p3 accel; or,
 p1 distance, p2 timeint, p3 speed; or,
 p1 angle, p2 timeint, p3 angrate; or,
 p1 speed, p2 accel, p3 timeint; or,
 p1 distance, p2 speed, p3 timeint; or,
 p1 angle, p2 angrate, p3 timeint; or,
 p1 displacement, p2 timeint, p3 velocity; or,
 p1 velocity, p2 timeint, p3 accel-vec;

p4 (if supplied) same type as p3
 p5 (if supplied) same type as p3

*Equality is defined in EC.DATA.2.6.1.

Program Effects

Any result of type angle is taken modulo 360 degrees.

- +MUL+ For multiplying a velocity (accel-vec) by a timeint: p3 = a displacement (velocity) vector with the same direction as the velocity (accel-vec), whose magnitude is equal to the timeint multiplied by the magnitude of the velocity (accel-vec).
- +DIV+ For dividing a displacement (velocity) by a timeint: p3 = a velocity (accel-vec) vector with the same direction as the displacement, whose magnitude is equal to the magnitude of the displacement (velocity) divided by the timeint.

Program effects for all other cases are defined in EC.DATA.2.6.2.

AT.NUM.2.6.3 Operations Converting from AT-Provided Scalar Types to Reals

<u>Program name</u>	<u>Parm type</u>	<u>Parm info</u>	<u>Undesired events</u>
+R_ACCEL_FPSS+ +R_ACCEL_G+	p1:accel;I p2:real;O	source destination	%range exceeded%
+R_ANGLE_DEG+ +R_ANGLE_CIR+ +R_ANGLE_RAD+ +R_ANGLE_SIN+ +R_ANGLE_COS+	p1:angle;I p2:real;O	source destination	
+R_ANGRATE_DEGHOUR+ +R_ANGRATE_RADSEC+	p1:angrate;I p2:real;O	source destination	
+R_DENSITY_SLCUFT+	p1:density;I p2:real;O	source destination	
+R_DISTANCE_FT+ +R_DISTANCE_NMI+	p1:distance;I p2:real;O	source destination	
+R_PRESSURE_INHG+	p1:pressure;I p2:real;O	source destination	%range exceeded%
+R_SPEED_FPS+ +R_SPEED_FPM+ +R_SPEED_KNOT+	p1:speed;I p2:real;O	source destination	

Program Effects

Each program provides a real value in p2 giving the physical quantity of p1 in the units abbreviated in the name of the program. The unit abbreviations are define in Table AT.NUM.a.

AT.NUM.2.6.4 Operations Converting from Reals to AT-Provided Scalar Types

<u>Program name</u>	<u>Parm type</u>	<u>Parm info</u>	<u>Undesired events</u>
+ACCEL_R_FPSS+ +ACCEL_R_G+	p1:real;I p2:accel;O	source destination	%range exceeded%
+ANGLE_R_DEG+ +ANGLE_R_CIR+ +ANGLE_R_RAD+	p1:real;I p2:angle;O	source destination	
+ANGLE_R_SIN+ +ANGLE_R_COS+	p1:real;I p2:angle;O	source destination	%illegal sin or cos given%
+ANGLE_R_SINCOS+	p1:real;I p2:real;I p3:angle;O	sine of angle cosine of angle destination	%range exceeded%
+ANGRATE_R_DEGHOUR+ +ANGRATE_R_RADSEC+	p1:real;I p2:angrate;O	source destination	
+DENSITY_R_SLCUFT+	p1:real;I p2:density;O	source destination	
+DISTANCE_R_FT+ +DISTANCE_R_NMI+	p1:real;I p2:distance;O	source destination	
+PRESSURE_R_INHG+	p1:real;I p2:pressure;O	source destination	
+SPEED_R_FPS+ +SPEED_R_FPM+ +SPEED_R_KNOT+	p1:real;I p2:speed;O	source destination	

Program Effects

- +ANGLE_R_SINCOS+ Produces in p3 the angle whose sine and cosine are equivalent to p1 and p2, respectively.
- +ANGLE_R_COS+ Produces in p2 the angle between 0 degrees and 180 degrees, inclusive, whose cosine is p1.
- +ANGLE_R_SIN+ Produces in p2 the angle between -90 degrees and +90 degrees, inclusive, whose sine is p1.

All other programs produce an AT-type value in p2 equivalent to p1, assuming that p1 represents a physical quantity in the units abbreviated in the program name. The unit abbreviations are defined in Table AT.NUM.a. Results of type angle are given modulo 360 degrees.

AT.NUM.2.6.5 Operations Converting from Vector Types

<u>Program name</u>	<u>Parm type</u>	<u>Parm info</u>	<u>Undesired events</u>
+XYZ_VECTOR+	p1:____;I p2:____;O p3:____;O p4:____;O	!+vector+! !+X component+! !+Y component+! !+Z component+!	%range exceeded%
+SPHER_VECTOR+	p1:____;I p2:____;O p3:angle;O p4:angle;O	!+vector+! !+magnitude+! !+theta+! !+phi+!	
+CYL_VECTOR+	p1:____;I p2:____;O p3:angle;O p4:____;O	!+vector+! !+radius+! !+theta+! !+Z component+!	

Parameter Requirements

+XYZ_VECTOR+	p1 velocity, and p2, p3, p4 speed; or, p1 displacement, and p2, p3, p4 distance; or, p1 accel-vec, and p2, p3, p4 accel
+SPHER_VECTOR+	p1 velocity, and p2 speed; or, p1 displacement, and p2 distance; or, p1 accel-vec, and p2 accel
+CYL_VECTOR+	p1 velocity, and p2 and p4 speed; or, p1 displacement, and p2 and p4 distance; or, p1 accel-vec, and p2 and p4 accel

Program Effects

All programs produce component equivalents from the given vector. No coordinate system frame of reference is assumed by this module; the frame of reference that users employed to initialize the vector will be the one to which the components correspond.

+XYZ_VECTOR+	p2, p3, and p4 are set to the x, y, and z Cartesian components (respectively) of the vector given by p1.
+SPHER_VECTOR+	p2, p3, and p4 are set to the spherical coordinate components of the vector given by p1.
+CYL_VECTOR+	p2, p3, and p4 are set to the cylindrical coordinate components of the vector given by p1.

AT.NUM.2.6.6 Operations Converting to Vector Types

<u>Program name</u>	<u>Parm type</u>	<u>Parm info</u>	<u>Undesired events</u>
+VECTOR_XYZ+	p1:____;I p2:____;I p3:____;I p4:____;O	!+X component+! !+Y component+! !+Z component+! !+vector+!	%range exceeded%
+VECTOR_SPHER+	p1:____;I p2:angle;I p3:angle;I p4:____;O	!+magnitude+! !+theta+! !+phi+! !+vector+!	
+VECTOR_CYL+	p1:____;I p2:angle;I p3:____;I p4:____;O	!+radius+! !+theta+! !+Z component+! !+vector+!	

Parameter requirements

+VECTOR_XYZ+	p1, p2, p3 speed and p4 velocity; or, p1, p2, p3 distance and p4 displacement; or, p1, p2, p3 accel and p4 accel-vec
+VECTOR_SPHER+	p1 speed and p4 velocity; or, p1 distance and p4 displacement; or, p1 accel and p4 accel-vec
+VECTOR_CYL+	p1 and p3 speed, and p4 velocity; or, p1 and p3 distance, and p4 displacement; or, p1 and p3 accel, and p4 accel-vec

Program Effects

+VECTOR_XYZ+	p4 = the vector equivalent of p1, p2, and p3, assuming a Cartesian coordinate system.
+VECTOR_SPHER+	p4 = the vector equivalent of p1, p2, and p3, assuming a spherical coordinate system.
+VECTOR_CYL+	p4 = the vector equivalent of p1, p2, and p3, assuming a cylindrical coordinate system.

AT.NUM.3 Undesired Event Assumptions

See EC.DATA.3. All Undesired Event assumptions there apply, except for those that explicitly mention bitstrings, registers, or time intervals. In addition:

1. User programs will not provide a real with magnitude greater than 1 to represent a sine or a cosine of an angle.

AT.NUM.4 Local Type Definitions

Local types used in the description of programs specified in EC.DATA are defined in Section EC.DATA.4, except for the following:

attribute	An ordered triple of numeric entities specifying lower bound, upper bound, and resolution.
type class	Enumerated: accel, accel-vec, angle, angrate, density, displacement, distance, pressure, speed, or velocity.
version	A version name applicable to the given type class. Version names and characteristics are listed in Appendix 5.

AT.NUM.5 Dictionary

Dictionary terms used to described programs specified in EC.DATA are defined in Section EC.DATA.5. The following terms are introduced by this module:

!+magnitude+!	The scalar magnitude of the vector; the distance between the end of the vector and the origin of the coordinate system.
!+phi+!	In spherical coordinates, the angle between the vector and positive z axis; sign of the angle is the sign of the vector's z component.
!+radius+!	The distance from the origin to the end of the projection into the x-y plane of the vector.
!+theta+!	The angle from the positive x axis to the projection of the vector into the x-y plane; the angle is measured counterclockwise as seen looking into the x-y plane in the negative z direction.
!+vector+!	The vector type class equivalent of the given components.
!+X component+!	The x component of the given vector.
!+Y component+!	The y component of the given vector.
!+Z component+!	The z component of the given vector.

AT.NUM.6 Undesired Event Dictionary

Undesired events described in programs specified in EC.DATA.2 are defined in Section EC.DATA.6. In addition:

%range exceeded% Defined in EC.DATA.6

%illegal sin or cos given% User has supplied a sine or cosine with
magnitude greater than 1.

AT.NUM.7 System Generation Parameters

None

AT.NUM.8 Information Hidden

1. The representation of numeric objects.
2. How range and resolution information are used to determine representation.
3. The procedures for performing numeric operations.
4. Conversions required, if two objects of the same type or type class are not represented in the same way.

AT.STE STATE TRANSITION EVENT TYPE CLASS

AT.STE.1 Introduction

AT.STE.1.1 Overview

This module implements State Transition Event (STE) types. Users may define specific types of this type class and declare entities of those types.

Each STE type is a class of equivalent finite state machines. The value of an STE variable is its state. Each STE type is characterized by a set of states, named subsets of that set, named relations on that set, and state transition events. It is intended that this module be used only when the number of states is small enough that the attributes of the type can be described by enumeration.

The relations may be used either to specify conditions relating two entities of the same STE type or to describe state transitions.

This module provides facilities that allow a process to await specified conditions and transitions. Awaiting a state transition event suspends the process until the event occurs. Awaiting a condition suspends the process until the condition holds.

AT.STE.1.2 Literals

Literals are state names and must begin and end with the character "\$".

AT.STE.2 Interface Overview

AT.STE.2.1 Declaration of Specific Types

STE types are declared by use of the form of ++DCL_TYPE++ (see EC.DATA.2.1). The forms of the parameters for STE type declarations are given below.

<u>Parameter</u>	<u>Type</u>	<u>Comments</u>
p1	name	any unused name
p2	type class	"STE"
p3	attributes	see AT.STE.4
p4	binding	all STE attributes must be fixed
p5	implementation	omitted

Program Effects

In addition to defining a type, the STE ++DCL_TYPE++ operation causes a number of access programs to be defined on elements of the type being declared. The syntax and semantics of these programs are discussed in AT.STE.2.5.

The following additional undesired events can occur in the declaration of STE types:

```
%%duplicate set member%%
%%malformed attribute%%
%%missing state attribute%%
%%too many state attributes%%
%%type inconsistency%%
%%undeclared spectype%%
%%unknown state%%
```

AT.STE.2.2 Declaration of Variables and Constants

STE variables and constants are defined by using the form of ++DCL_ENTITY++ (see EC.DATA.2.2). Parameter p3 (initial attribute) is omitted for STE type variables.

AT.STE.2.3 Declaration of Arrays

Arrays of STE type entities are defined by using the form of ++DCL_ARRAY++ (see EC.DATA.2.3). Parameter p3 (initial attribute) is omitted for STE type variables.

AT.STE.2.4 Operand Descriptions

See description in EC.DATA.2.4. Those portions of the description in EC.DATA.2.4 pertaining to variables with varying attributes, and to registers, do not apply to this module.

The following additional undesired event can occur when operands are specified.

%%inappropriate parameters%%

AT.STE.2.5 Access Programs

The declaration of an STE type defines a number of access programs that operate on entities of that type. For each attribute identifier that appears in a "conversion", "relation", or "set" attribute declaration, an access program is defined having that identifier as its name. In addition, several additional access programs are defined whose names are systematically derived from the attribute identifiers. For a given STE type declaration, the names of the access programs that are defined by that declaration may be found by systematically replacing the strings 'set', 'relation', 'conversion1', and 'conversion2' with the respective identifier of each set, relation, or conversion declaration of that type.

The following additional undesired event can occur when an STE access program is invoked.

%%unknown STE program%%

AT.STE.2.5.1 Operations on the STE Entities

<u>Program name</u>	<u>Parm type</u>	<u>Parm info</u>	<u>Undesired events</u>
+ 'set' +	p1:STETYPE;I p2:boolean;O	entity to test destination	
+IS_ 'relation' +	p1,p2:STETYPE;I p4:(parms);I_OPT p3:boolean;O	pair to test n-tuple destination	%%unequal lists%%
+ 'relation' +	p1:STETYPE;O p2:(parms);I_OPT	domain elements n-tuple	%%unequal lists%% %%out of domain%
+ 'conversion1' +	p1:STETYPE;I p2:convtype;O	value to convert destination	
+ 'conversion2' +	p1:convtype;I p2:STETYPE;O	value to convert destination	

Program Effects

- + 'set' + $p2 := (p1 \text{ is an element of the set attribute whose identifier is 'set'})$

- + IS_ 'relation' + $p3 := ((p4, p1), p2) \text{ is an element of the relation attribute whose identifier is 'relation'}. \text{ If } p4 \text{ is omitted, then } p3 := (p1, p2) \text{ is an element of 'relation'}.$

- + 'relation' + $p1 \text{ changes state to } s2 \text{ such that } s1 \text{ is the value of } p1 \text{ before the call, } v1, \dots, vn \text{ is the value of } p2 \text{ (if given), and the ordered pair } ((v1, \dots, vn), s1), s2) \text{ (or } (s1, s2) \text{ where } p2 \text{ is omitted) is an element of the relation attribute whose identifier is 'relation'}. \text{ Where there is more than one ordered pair with the same first element, any may be chosen.}$

- + 'conversion1' + $p2 := x \text{ such that } (p1, x) \text{ is an element of a conversion relation, and 'conversion1' is the first of the ordered pair of identifiers associated with that conversion definition.}$

- + 'conversion2' + $p2 := x \text{ such that } (x, p1) \text{ is an element of a conversion relation, and 'conversion2' is the second of the ordered pair of identifiers associated with that conversion definition.}$

AT.STE.2.5.2 Await Operations

In this section "AWAIT.T/F.'string'" stands for two names, "AWAIT.T.'string'" and "AWAIT.F.'string'". Similarly, in the effects section, alternate phrasing for each member of the pair is separated by "/".

<u>Program name</u>	<u>Parm type</u>	<u>Parm info</u>	<u>Undesired events</u>
+AWAIT@'relation'+	$p1:ST\text{Etype};I$ $p2:(\text{parms});I_OPT$	variable to watch n-tuple	
+AWAIT@='relation'+	$p1:ST\text{Etype};I$ $p2:(\text{parms});I_OPT$	variable to watch n-tuple	
+AWAIT.T/F.'set'+	$p1:ST\text{Etype};I$	candidate member	
+AWAIT.T/F.'relation'+	$p1, p2:ST\text{Etype};I$ $p3:(\text{parms});I_OPT$	candidate pair n-tuple	

Program Effects

- +AWAIT@'relation'+ Let $p2'$ be the value of parameter(s) $p2$. Then, all action in the calling process is suspended until $p1$ undergoes a state change from $s1$ to $s2$ such that the ordered pair $((p2', s1), s2)$ (or $(s1, s2)$ if $p2$ is omitted) is an element of the relation attribute whose identifier is 'relation'.

- +AWAIT@='relation'+ If p2 is omitted, then all action in the calling process is suspended until the access program +'relation'+ (p1) is executed. Otherwise, let p2' be the value of p2. Then all action in the calling process is suspended until the access program +'relation'+ (p1,p2) is executed with p2=p2'.
- +AWAIT.T/F.'set'+ If the value of p1 is/(is not) a member of the set attribute whose identifier is 'set', the call has no effect. Otherwise, all action in the calling process is suspended until the value of p1 is/(is not) a member of 'set'.
- +AWAIT.T/F.'relation'+ Let p3' be the value of parameter(s) p3. Then, all action in the calling process is suspended until the ordered pair of values given by ((p3',p1),p2) (or (p1,p2) if p3 is omitted) is/(is not) a member of the relation attribute whose identifier is 'relation'. If the ordered pair of values is/(is not) already a member of 'relation', the call has no effect.

AT.STE.3 Undesired Event Assumptions

1. Users will not supply an argument to a conversion outside of the domain of the defining relation.

AT.STE.4 Local Type Definitions

Local types that are used in the description of programs specified in EC.DATA and are not described here are defined in Section EC.DATA.4. In the following definitions, where the syntax is given in modified BNF, alphabetic strings contained in quotation marks (e.g., "state") are terminal symbols. Ellipses (e.g., state, . . . , state) denote a list of two or more elements separated by commas.

- attribute The attributes of an STE type characterize the type. The syntax of an STE type attribute is,
 ::= ("state" (state_list)
 or ("relation" (relation_defn))
 or ("set" (set_defn))
 or ("conversion" (conversion_defn))
- An STE type declaration must have exactly one "state" attribute. There may be any number of the remaining attributes.
- attributes ::= attribute or attribute, attributes.
- convtype An entity of the specific type specified in the conversion_defn.
- conversion_defn A 1:1 mapping between a subset of the states of an STE type and a set of literals of another type. The syntax is,

$::= (ID, ID), type_ID (s_1, t_1)$
 $\quad \underline{or} (ID, ID), type_ID ((s_1, t_1), \dots, (s_n, t_n))$

where $type_ID$ is the identifier of a specific type. each of the subsequent ordered pairs must be of the form:

$(STE\ literal, x)$ or $((STE\ literal, \dots, STE\ literal), x)$

where x is a literal or constant of $type_ID$. The ordered pair (ID, ID) identifies the conversion programs from the STE type and to the STE type, respectively.

ID	An identifier associated with an STE attribute.
parms	A list of entities separated by commas. The number of entities and their types must match those of the n-tuple "P" in the corresponding relation_defn.
relation_defn	<p>A set of ordered pairs of the form $((P, D), R)$ where (P, D) is an element of the domain of the relation being defined and R is an element of the range. D and R must be m-tuples of elements of the state set. P may be an n-tuple of literals of any type. P may be omitted in which case, the ordered pairs are written (D, R). The syntax is,</p> <p>$::= ID (ordered_pairs)$</p> <p>$ordered_pairs ::= (domain, state_list)$ $\quad \underline{or} (domain, state_list), \dots, (domain, state_list)$</p> <p>$domain ::= state_list$ $\quad \underline{or} (literals, state_list)$</p> <p>$literals ::= literal$ $\quad \underline{or} (literal, \dots, literal)$</p> <p>The relation "EQ" is automatically defined for all STE types. It is defined as the set of ordered pairs (s, s) where s is an n-tuple of elements of the state set.</p> <p>The relation "SET" is automatically defined for all STE types. It is defined as the set of all ordered pair $((s_2, s_1), s_2)$ where s_1 and s_2 are lists of elements of the state set.</p>
set_defn	<p>A subset of the set of states of the STE type being defined or a subset of the power set of the set of states. The syntax is,</p> <p>$::= ID, state_list$ $\quad \underline{or} ID (state_list, \dots, state_list).$</p>
state	An STE literal (i.e., state name).

state_list ::= state or (state, . . . , state)

STE type An entity of the STE type for which the operation is defined or a list of such entities enclosed in parentheses and separated by commas.

AT.STE.5 Dictionary

None.

AT.STE.6 Undesired Event Dictionary

%%duplicate set member%%	A user program has specified a set with duplicate elements.
%%inappropriate parameters%%	A user program has called an access program with parameters that do not correspond in specific type to the declaration.
%%malformed attribute%%	The user has supplied an attribute declaration with a syntax inconsistent with that specified in AT.STE.4.
%%missing state attribute%%	A user program has declared an STE type with no "state" attribute.
%out of domain%	A user program has supplied an input parameter that is not in the domain of the relation.
%%too many state attributes%%	A user program has declared an STE type with more than one "state" attribute.
%%type inconsistency%%	A user program has supplied a conversion value not of the specified type.
%%undeclared spectype%%	A user program has specified a specific type that has not been defined.
%%unequal lists%%	A user program has supplied a list parameter that does not match the specified length.
%%unknown state%%	A user program has provided a state value that has not been declared in the "state" attribute.
%%unknown STE program%%	A user program has specified an STE program that does not correspond to any attribute identifier that has been defined for the type.

AT.STE.7 System Generation Parameters

None.

AT.STE.8 Information Hidden

1. Internal representation of states, sets, and relations.
2. Algorithms used in the programs corresponding to AT attributes.
3. How processes await a state transition event and how they are restarted.

AT.INDEX INDICES TO THE DOCUMENT

This section provides the following indices to the facilities described in this document:

Access Programs

Local Type Definitions

Dictionary Terms

Undesired Events

Facilities specified in [EC] are not included in this index.

Access Programs

<u>Program name</u>	<u>Where defined</u>
+ACCEL_R_FPSS+	AT.NUM
+ACCEL_R_G+	AT.NUM
+ANGLE_R_CIR+	AT.NUM
+ANGLE_R_COS+	AT.NUM
+ANGLE_R_DEG+	AT.NUM
+ANGLE_R_RAD+	AT.NUM
+ANGLE_R_SIN+	AT.NUM
+ANGLE_R_SINCOS+	AT.NUM
+ANGRATE_R_DEGHOURL+	AT.NUM
+ANGRATE_R_RADSEC+	AT.NUM
+CYL_VECTOR+	AT.NUM
+DENSITY_R_SLCUFT+	AT.NUM
+DISTANCE_R_FT+	AT.NUM
+DISTANCE_R_NMI+	AT.NUM
+PRESSURE_R_INHG+	AT.NUM
+R_ACCEL_FPSS+	AT.NUM
+R_ACCEL_G+	AT.NUM
+R_ANGLE_CIR+	AT.NUM
+R_ANGLE_COS+	AT.NUM
+R_ANGLE_DEG+	AT.NUM
+R_ANGLE_RAD+	AT.NUM
+R_ANGLE_SIN+	AT.NUM
+R_ANGRATE_DEGHOURL+	AT.NUM
+R_ANGRATE_RADSEC+	AT.NUM
+R_DENSITY_SLCUFT+	AT.NUM
+R_DISTANCE_FT+	AT.NUM
+R_DISTANCE_NMI+	AT.NUM
+R_PRESSURE_INHG+	AT.NUM
+R_SPEED_FPM+	AT.NUM
+R_SPEED_FPS+	AT.NUM
+R_SPEED_KNOT+	AT.NUM
+SPEED_R_FPM+	AT.NUM
+SPEED_R_FPS+	AT.NUM

+SPEED_R_KNOT+	AT.NUM
+SPHER_VECTOR+	AT.NUM
+VECTOR_CYL+	AT.NUM
+VECTOR_SPHER+	AT.NUM
+VECTOR_XYZ+	AT.NUM
+XYZ_VECTOR+	AT.NUM

In addition, AT.STE allows the user to define access programs of the following form:

```
+AWAIT@relation+
+AWAIT@=relation+
+AWAIT.T/F.set+
+AWAIT.T/F.relation+
+conversion1+
+conversion2+
+relation+ (4 parameters)
+relation+ (2 parameters)
+set+
```

Local Type Definitions

<u>Type name</u>	<u>Where defined</u>
accel	AT.NUM
accel-vec	AT.NUM
angle	AT.NUM
angrate	AT.NUM
attribute	AT.NUM, AT.STE
attributes	AT.STE
convtype	AT.STE
conversion_defn	AT.STE
density	AT.NUM
displacement	AT.NUM
distance	AT.NUM
parms	AT.STE
pressure	AT.NUM
relation_defn	AT.STE
set_defn	AT.STE
speed	AT.NUM
state_set	AT.STE
STEtype	AT.STE
STEtype_list	AT.STE
typeclass	AT.NUM
velocity	AT.NUM
version	AT.NUM

Dictionary Terms

<u>Term</u>	<u>Where defined</u>
!+magnitude+!	AT.NUM
!+phi+!	AT.NUM
!+radius+!	AT.NUM
!+theta+!	AT.NUM

!+vector+!	AT.NUM
!+X component+!	AT.NUM
!+Y component+!	AT.NUM
!+Z component+!	AT.NUM

Undesired Events

<u>UE name</u>	<u>Where defined</u>
%%duplicated set member%%	AT.STE
%illegal sin or cos given%	AT.NUM
%%inappropriate parameters%%	AT.STE
%%malformed attribute%%	AT.STE
%%missing state attribute%%	AT.STE
%out of domain%	AT.STE
%range exceeded%	AT.NUM
%%too many state attributes%%	AT.STE
%%type inconsistency%%	AT.STE
%%undeclared spectype%%	AT.STE
%%unequal lists%%	AT.STE
%%unimplemented binding%%	Appendix 4
%%unimplemented attribute via variables%%	Appendix 4
%%unknown state%%	AT.STE
%%unknown STE program%%	AT.STE

ACKNOWLEDGMENTS

The authors gratefully acknowledge the following people who reviewed this document for consistency, clarity, and correctness:

Ms. Dawn Janney	Naval Weapons Center, China Lake, CA
Ms. Jo Miller	
Dr. Don Utter	Bell Laboratories, Columbus, OH
Mr. Glenn Cooper	Vought Corporation, Dallas, TX
Prof. Michael Levy	Department of Computer Science University of Victoria Victoria, British Columbia

REFERENCES

- [DI] Parker, Heninger, Parnas, and Shore, *Abstract Interface Specifications for the A-7E Device Interface Module*; NRL Memorandum Report 4385; November 1980.
- [EC] Britton, Clements, Parnas, and Weiss; *Interface Specifications for the SCR (A-7E) Extended Computer Module*; NRL Memorandum Report 4843; May 1982.
- [SO] Software Cost Reduction Project; *A Standard Organization for Abstract Interface Specifications*; NRL Memorandum Report in progress. Until publication, readers are referred to the "Standard Organization," chapter of [DI] instead.

Appendix 1
INTERFACE DESIGN ISSUES

AT.NUM

1. Although AT and EC designers are critically concerned with which facilities are machine-dependent and which are not, we felt that users of the resulting modules would not be as concerned. Therefore, we tried to design this module so that users could regard it as an extension to the Extended Computer; the AT and EC in conjunction provide the basic programming environment for the rest of the system. We tried to make the design as parallel to EC.DATA as possible, in order to present in effect a single kind of interface to programmers interested in manipulating data of any type.
2. We thought that registers for abstract data types would not be useful. If they turn out to be desirable, however, the facility would be a straightforward extension, based on the EC architecture.
3. Any design issue in EC.DATA concerning facilities or designs copied by this module naturally also applies to this module.
4. We could have allowed a sine/cosine pair for an angle literal. However, to keep the interface simple and consistent, we do not allow literals to be given in this two-part manner.
5. We used to have an abstract data type called "mach", using it to represent a speed relative to the speed of sound. But it produced so many hard questions (e.g., When you divide a speed by a speed, do you get a real or a mach? How can you check at compile-time? If the speed you compare with is the speed of sound at some other altitude/pressure/temperature, is the result a mach or not?) that we decided it was not really an abstract data type like the others.

AT.STE

1. Originally, the AT module included only the synchronization operations `+SIGNAL+` for signaling the occurrence of events and `+AWAIT+` for allowing processes to wait for the occurrence of events. These operations were chosen because they support the notion of events in the OFP specifications.

These operations originally were part of the EC interface. They were moved to the AT module because (a) they can be constructed by using the `+UP+`, `+DOWN+`, and `+PASS+` operations provided by the EC; (b) the EC synchronization operations would not be made simpler by using `+AWAIT+` and `+SIGNAL+`; (c) we could think of useful systems that did not need `+AWAIT+` and `+SIGNAL+`; and (d) it made the EC interface simpler.

2. It was noted that, in some cases, processes needed to wait if the system was not in a particular state but did not need to wait if the the system was already in the desired state. (Here and in the following paragraphs, "state of the system" is used to mean a particular system state or a class of such states.) Simple event variables could not be used in this case since the event associated with the system entering a state would not be signaled until the state had been left and entered again. To solve this problem, we added an additional event variable to the interface called an "event

boolean" and an additional await operator. Processes could wait on the event of a state change in an event boolean using the +AWAIT+ operator. The new operator, +AWAITC+ (await on condition), caused the process to wait only if the event boolean was not in the specified state. If the event boolean was already in the specified state, the calling process continued without interruption. This allowed processes to respond to an event that occurred while the process was active. This facility was later superseded (see 4).

3. The specifications require that processes be able to wait on the occurrence of complex events, for instance, the disjunction of two or more events or the occurrence of an event while the system is in a particular state (i.e., @T(event) WHEN (condition AND condition AND . . .)). Complex synchronization conditions could not be implemented directly with even booleans, so we attempted to provide a syntax for expressing complex events and a synchronization mechanism for interpreting that syntax. Users of the synchronization module would express the complex event for which they wanted to wait in the syntax provided by this module. The module would interpret the requests, translate them into more primitive synchronization operators, and signal the appropriate events.

This scheme proved inadequate for several reasons. The synchronization module proved to be large and complicated. The interpretation mechanism was complex and difficult to implement. The proposed mechanism still did not solve all of the problems associated with waiting for complex events:

- a. If user processes were awaiting a disjunction of events, they could not distinguish which event had been signaled. Processes could not perform conditional actions based on the awakening event without checking the values of extra variables. Even with very short deadlines, these values could change before they were checked.
- b. Processes waiting on events with WHEN clauses frequently needed to know if the conditions expressed in the WHEN clause still held by the time they began running. These processes would have to recheck the condition values after they began running.
- c. Since the events that processes awaited did not necessarily reflect the state of the system by the time they began running, the order in which these events had occurred could not necessarily be determined. Processes that needed to run in a particular order depending on the order of events were not able to do so.

By using even variables and event booleans, these problems could be solved only by providing great numbers of small short-deadline processes that recorded the occurrences of events in local variables.

4. STE variables were proposed as a replacement for event variables and event booleans. Users can define event variables with more than two states and can wait on state change events or transitions between states. This mechanism allows us to solve those problems listed in 3, where the class of states is small enough to describe by enumeration and the transition time between states is large enough that the changes can be recorded by the event detection mechanism; for instance:
 - a. Event booleans are unnecessary since they can be implemented with STEs.
 - b. STEs provide a single mechanism for signaling transitions in variables with more than two discrete states.

- c. A syntax for expressing awaits on complex events is unnecessary. A state of an STE variable can be associated with the occurrence of a complex event, and the user process need only wait on the transition to that state.
 - d. User processes can wait on a disjunction of events and determine which event caused the signal by examining the value of an STE variable.
 - e. STEs can record the "history" of events in their transitions. In those cases where the history can be represented by an enumerated set of states and transitions, STEs allow the user processes to respond to events in the order of their occurrence and to events that have occurred while they were running. Additional processes need not be created to perform these tasks.
 - f. Processes can signal different events to different users by causing a transition in a single STE variable.
5. Originally, the STE module provided comparison operators for STE values and set membership operators and conversion operators. We found that users of the module frequently only needed a small subset of the operations provided. By allowing the creator of the STE type to define the operations needed, the required subset of operations can be selected without incurring overhead for unneeded ones.
6. Originally the STE module provided for one explicit ordering on a given STE domain. The comparison operators provided referred to that ordering. We decided that such an ordering was unnecessary as it can be implemented as a special case of a relation.

Appendix 2
IMPLEMENTATION NOTES

AT.NUM

None.

AT.STE

None.

Appendix 3

BASIC ASSUMPTIONS

AT.NUM

All basic assumptions in EC.DATA.3.1 apply, except those that explicitly mention bitstrings, registers, or time intervals. Where an EC.DATA assumption mentions the Extended Computer or EC, readers of this module should substitute "Application Data Types module" or "AT," respectively. In addition, the following assumptions apply to this module:

1. For each scalar type class provided, there is a fixed set of units of measurement into which values of that class may be converted. The units for each type class are listed in Table AT.NUM.a. Values of vector type classes may be converted into component scalar values, or a direction/magnitude equivalent.
2. User programs may not make assumptions about the representation of numeric values. Even though literals are expressed in particular units, there is no implication that the value is stored in those units.
3. The only operations needed for calculating new numeric values are: addition, multiplication, division, subtraction, absolute value, and complement. In addition, we need to convert scalars to/from reals, and vectors to/from component scalars.
4. We need arrays of numeric data types in which the attributes of an element can change independently of the attributes of other elements.
5. The range and resolution of each numeric variable can be determined at the time the system is generated.
6. Arithmetic operations involving operands of different type classes may or may not have a useful physical meaning. Those that do are: $\text{speed} = \text{distance}/\text{time}$, $\text{distance} = \text{speed} \cdot \text{time}$, $\text{time} = \text{distance}/\text{speed}$, $\text{angrate} = \text{angle}/\text{time}$, $\text{angle} = \text{angrate} \cdot \text{time}$, $\text{time} = \text{angle}/\text{angrate}$, $\text{accel} = \text{speed}/\text{time}$, $\text{speed} = \text{accel} \cdot \text{time}$, $\text{time} = \text{speed}/\text{accel}$, $\text{velocity} = \text{displ}/\text{time}$, $\text{displ} = \text{velocity} \cdot \text{time}$, $\text{time} = \text{displ}/\text{velocity}$, $\text{accel-vec} = \text{velocity}/\text{time}$, $\text{velocity} = \text{accel-vec} \cdot \text{time}$, and $\text{time} = \text{velocity}/\text{accel-vec}$.

Further, if a value of any type class is multiplied or divided by a real, the result has the same type class.

AT.STE

1. State transitions may be considered to be instantaneous.
2. When processes need to wait for particular states to occur, they do not need to proceed while waiting. The states for which the process is waiting can be determined before the process begins to wait.

3. If a process is awaiting a state change, it need not proceed until the change occurs. The transitions for which the process is waiting can be determined before the process begins to wait.
4. There is no need for a mechanism that allows a process to be started before the state or transition that has been specified in an AWAIT operation has occurred.

Appendix 4

UNIMPLEMENTED APPLICATION DATA TYPE FACILITIES

Not all of the capabilities described in this report have been provided in the current version of the implementation. A few facilities, which are not currently needed by the application program, have not been implemented. An attempt to use an absent facility will result in an undesired event in the development version. The unimplemented features are described below.

- Feature:** Specific types with attributes that can vary at run-time
- Where Described:** AT.NUM.
- Undesired Event:** %%unimplemented binding%%.
- Current Use:** In the ++DCL_TYPE++ program, users may not declare the binding of bitstring or timeint specific types to be VARY. In the ++DCL_ENTITY++ and ++DCL_ARRAY++ programs, users may not provide an initial attribute.
-
- Feature:** Using variables to specify attributes of a specific type, or of a variable or array with varying attributes.
- Where Described:** AT.NUM.
- Undesired Event:** %%unimplemented attribute via variables%%.
- Current Use:** To specify an attribute (as defined in AT.NUM.4), literals or constants must be used.

Appendix 5

DATA REPRESENTATION (VERSION) CATALOG

For some numeric data types, the Application Data Types module can provide more than one kind of representation. The version has no effect on the outcome of an operation, but some versions allow some operations to be performed more quickly than other versions.

The version catalog lists the provided version names for each AT data type which has more than one version. When declaring a specific type, users may request a particular version by using these names.

Users referred to the version catalog in Appendix 6 of [EC]. Versions of real types named there are available from this module as well.