(4)

Technical Document 1944
December 1990

User's Guide to an Event-Activation Record Approach to Simulation Modeling in Ada

H. Mumm
R. Ollerton

ADMINISTRATIVE INFORMATION

This report was completed under funding from the Naval Ocean Systems Center for the Shared Adaptive Internetworking Project (SAINT) as part of the Communication Block Program.

The need for an affordable and understandable discrete-event simulation system became apparent during earlier work. The current work demonstrated that Ada was not only adequate, but was in many ways superior to currently available commercial simulation languages.

Released by
M. B. Vineberg, Head
System Design and Architecture
Branch

Under authority of
K. R. Casey, Head
Battleforce and Theater
Communications Division

CONTENTS

1.0 INTRODUCTION

This document explains how to install and use the Event-Activation Record Approach to Simulation Modeling in Ada (EARS) simulation libraries, Version 1.0, that have been developed by the Naval Ocean Systems Center (NOSC), Code 854. EARS, Version 1.0, resides on a NOSC VAX 8600 named "Otter" and executes using the VAX Ada, Version 2.0, compiler. EARS is a discrete-event simulation system that uses an event-activation record approach and has capabilities similar to those provided by SIMSCRIPT. All simulations that can be modeled in SIMSCRIPT can be modeled using EARS.

From a set of user-defined event types, EARS automatically generates much of the source code for many Ada packages and subprograms that are needed for simulation development. This allows a user to quickly start writing an EARS simulation. The user tailors the simulation to his or her application by adding Ada source code to the generated program units and by writing additional ones.

EARS packages contain subprograms for scheduling events, manipulating queues, calculating statistics, writing reports, debugging, and performing other necessary simulation functions.

This document includes an example of actual EARS simulation as well as examples of how to use EARS packages and subprograms.

Major differences between SIMSCRIPT and EARS include the following:

1. EARS does not implement processes; rather, events are used for synchronization.

2. Many constructs built by the SIMSCRIPT compiler must be constructed in EARS by using Ada statements. Advantages to using the Ada language rather than SIMSCRIPT include the following:

a. Ada supports software engineering features (e.g., strong typing, visibility, and generics).
b. Ada is the DoD standard programming language.
c. Ada symbolic debuggers greatly reduce debugging times.
d. Ada compilers are widely available (for many computers from many vendors).

This document assumes that the EARS user is already familiar with discrete-event simulation and knows how to use SIMSCRIPT. The EARS user must be an experienced Ada programmer.

2.0 INSTALLATION

2.1 CREATING THE EARS PROGRAM LIBRARY

EARS software is installed on a VAX Ada environment by creating an Ada library and then compiling the EARS files into the library. This may be done by executing the command file listed below. The EARS program library name, given below, was arbitrarily chosen.

```
$set verify                          ! Echos commands to monitor.
$acs create lib [320036.EV]          ! Creates ada program library
                                     ! [320036.EV].
$acs set lib [320036.EV]             ! Set to ada program library
                                     ! [320036.EV].
$ada/list/noopt SIM_.ADA             ! Compiles file sim_.ada, compiled code
                                     ! goes into SIM_.LIS file, and Ada
                                     ! identifiers not optimized away.
$ada/list/noopt RELATIONAL_.ADA
$ada/list/noopt RELATIONAL.ADA
$ada/list/noopt FLT_.ADA
$ada/list/noopt D2_.ADA
$ada/list/noopt DATE_.ADA
$ada/list/noopt STAT_.ADA
$ada/list/noopt AF_.ADA
$ada/list/noopt AI_.ADA
$ada/list/noopt TF_.ADA
$ada/list/noopt TI_.ADA
$ada/list/noopt GET_N_.ADA
$ada/list/noopt BIT_.ADA
$ada/list/noopt BIT.ADA
$ada/list/noopt HEX_.ADA
$ada/list/noopt HEX.ADA
$ada/list/noopt I_MAX_.ADA
$ada/list/noopt I_MAX.ADA
$ada/list/noopt I_MIN_.ADA
$ada/list/noopt I_MIN.ADA
$ada/list/noopt LINK_.ADA
$ada/list/noopt LINK.ADA
$ada/list/noopt LINK_AI_.ADA
$ada/list/noopt LINK_AI.ADA
$ada/list/noopt LINK_TI_.ADA
$ada/list/noopt LINK_TI.ADA
$ada/list/noopt LNK_.ADA
$ada/list/noopt LNK.ADA
$ada/list/noopt LNK_AI_.ADA
$ada/list/noopt LNK_AI.ADA
$ada/list/noopt HASH_.ADA
$ada/list/noopt HASH.ADA
$ada/list/noopt NAT_LNK.ada
$ada/list/noopt HSH_.ADA
$ada/list/noopt HSH.ADA
$ada/list/noopt PROTO_.ADA
$ada/list/noopt PROTO.ADA
$ada/list/noopt EV_.ADA
$ada/list/noopt EV_Q.ADA
$ada/list/noopt SQ_.ADA
$ada/list/noopt SQ.ADA
$ada/list/noopt DND_.ADA
$ada/list/noopt DND.ADA
$ada/list/noopt MTH_RANDOM_.ADA
$ada/list/noopt RAND_.ADA
$ada/list/noopt REAL_PARAM_.ADA
```

continued

```
$ada/list/noopt INT_PARAM_.ADA
$ada/list/noopt RANDOM_DISTRIBUTION_.ADA
$ada/list/noopt RANDOM_DISTRIBUTION.ADA
$ada/list/noopt RANDOM.ADA
$ada/list/noopt RAND.ADA
$ada/list/noopt SIM.ADA
$ada/list/noopt EV.ADA
$ada/list/noopt AF.ADA
$ada/list/noopt AI.ADA
$ada/list/noopt DATE.ADA
$ada/list/noopt GET_N.ADA
$ada/list/noopt INT_PARAM.ADA
$ada/list/noopt REAL_PARAM.ADA
$ada/list/noopt RL_AI.ADA
$ada/list/noopt STAT.ADA
$ada/list/noopt TF.ADA
$ada/list/noopt TI.ADA
$ada/list/noopt FLT.ADA
$set noverify                      ! Turn off echo to monitor.
```

2.2 CREATING THE GEN PROGRAM LIBRARY

The program library GEN, which contains the automatic source code generation software for both the EARS and the Simulated Network Architecture Protocol Layer (SNARPL) systems, is installed on a VAX Ada environment by executing the command file listed below. The program library name, GEN, was arbitrarily chosen.

```
$set verify                        ! Echos commands to monitor.
$acs create lib [320036.GEN]       ! Create program library
                                   ! [320036.GEN].
$acs set lib [320036.GEN]          ! Set to library just created.
$ada/list/noopt LC_.ADA
$ada/list/noopt MC_.ADA
$ada/list/noopt UC_.ADA
$ada/list/noopt NAME_.ADA
$ada/list/noopt FMT_.ADA
$ada/list/noopt ADA_NAME_.ADA
$ada/list/noopt ADA_NAME.ADA
$ada/list/noopt VMS_ACS_.ADA
$ada/list/noopt D2_.ADA
$ada/list/noopt DATE_.ADA
$ada/list/noopt DATE.ADA
$ada/list/noopt EV_GEN_.ADA
$ada/list/noopt EV_GEN.ADA
$ada/list/noopt GEN_SNARPL_GEN_.ADA
$ada/list/noopt GEN_SNARPL_GEN.ADA
$ada/list/noopt SNARPL_GEN.ADA
$ada/list/noopt FMT.ADA
$ada/list/noopt MC.ADA
$ada/list/noopt VMS_ACS.ADA
$ada/list/noopt UC.ADA
$ada/list/noopt NAME.ADA
$set noverify                      ! Turn off echo to monitor.
```

3.0 BUILDING AND RUNNING AN EARS SIMULATION

This chapter describes in detail how to write an EARS simulation. A complete simulation example is given. Included is an explanation of (1) how to generate the packages and subprograms automatically, (2) the code that is added by hand, and (3) the required inputs and outputs.

3.1 WRITING A PROCEDURE TO INSTANTIATE PACKAGE EV_GEN

The initial step of an EARS simulation is to instantiate the package Ev_Gen, which is in the GEN library. The Ev_Gen package specification is listed below.

```
with Text_IO, Ada_Name, Name, Vms_Acs, Date;
use Text_IO;

generic

  Sim_Name
    : STRING;
    --| ----------------------------------------------------------
    --| The first part of all file and compilation unit names
    --| will be "SIM_NAME".
    --| ----------------------------------------------------------

  type EVENTS is
    ( <> );
    --|----------------------------------------------------------
    --| Names of the different types of events initially in the
    --| simulation.
    --| ----------------------------------------------------------

  with procedure Simple_File_Name
    ( Unit : STRING;
      Specification : BOOLEAN;
      Fn : out Ada_Name.Name_String;
      Spec_Ext : STRING := Vms_Acs.Spec_Ext;
      Body_Ext : STRING := Vms_Acs.Body_Ext;
      Format : Name.String_Format := Name.Default_Case)
 is Vms_Acs.Simple_File_Name;
  --| -------------------------------------------------------------
  --| Form a file name for a unit specification or body.
  --| -------------------------------------------------------------
  with procedure File_Header
    ( File : File_Type;
      User_Name : STRING;
      Msg : STRING ) is Date.Header;
  --| -------------------------------------------------------------
  --| If USER_NAME is not a null string, this routine is called
  --| for each file created.
  --| -------------------------------------------------------------

procedure Ev_Gen
  ( User_Name : STRING := "";
    Spec_Ext : STRING := Vms_Acs.Spec_Ext;
    Body_Ext : STRING := Vms_Acs.Body_Ext;
    Format : Name.String_Format := Name.Default_Case);
```

The use of Ev_Gen will first be discussed in general terms. Ev_Gen is a generic package that may be instantiated with four parameters. The first two are required, while the last two can accept defaults. The first parameter, Sim_Name, is a string that becomes the first part of the file names and program unit names for the source code that is generated automatically. The second parameter, EVENTS, is an enumerated type that defines event types required for a specific simulation. Note, that the source code for a stop simulation event type will be generated automatically. It need not be specified in the enumerated type.

If Ev_Gen is not instantiated with the two user-provided procedures, Simple_File_Name and File_Header, then the default procedures are used. The default procedure Simple_File_Name creates VAX/VMS files and should be used when running EARS on VAX/VMS. The default procedure File_Header creates a file header.

An example of a default file header (supplied by package Date) that is created during code generation is given below. The user name, date, time, and a brief program unit description appear in the header.

```
--| -----------------------------------------------------
--| Alan, MONDAY 19-MAR-1990 14:56.48
-->>Pkg body for Simulation Job Definitions
--| -----------------------------------------------------
```

Two examples of procedures that instantiate Ev_Gen will now be described. The first, which is shown below, uses the default Simple_File_Name and File_Header procedures. This simulation defines four events: Message_Generation, Start_Slot, Transmit, and Receive (in addition to the stop simulation event). The first two characters of the file names and program unit names will start with the character string "MV."

```
with Ev_Gen;
procedure Gen_Source_MV is
  type Event_Type is (Message_Generation, Start_Slot,
                      Transmit, Receive);
  procedure EV is new Ev_Gen (Sim_Name =>"MV",
                              EVENTS => Event_Type);
begin
  EV(User_Name => "Alan");
end Gen_Source_MV;
```

For the second example, the user first writes the header procedure, SAIC_H, which is given below.

```
----------------------------------------------
-- SAIC_H will add standard headers to all files.
----------------------------------------------
with TEXT_IO;
use  TEXT_IO;

procedure SAIC_H (F          : file_type;
```

continued

```
                    USER_NAME : string;
                    MSG       : string) is
begin
put_line(f,"---------------------------------------------------------------------");
put_line(f,"--         xxxxxxx H F  ");
put_line(f,"---------------------------------------------------------------------");
put_line(f," ");
put_line(f,"---------------------------------------------------------------------");
put_line(f,"--         simulation: " & USER_NAME);
put_line(f,"--         module:     " & MSG);
put_line(f,"---------------------------------------------------------------------");
put_line(f," ");
put_line(f,"---------------------------------------------------------------------");
put_line(f,"-- ");
put_line(f,"---------------------------------------------------------------------");
put_line(f,"--         simulation: " & USER_NAME);
put_line(f,"--         module:     " & MSG);
put_line(f,"---------------------------------------------------------------------");
put_line(f," ");
put_line(f,"---------------------------------------------------------------------");
put_line(f,"-- ");
put_line(f,"-- Author:        E. Beck");
put_line(f,"-- Date:          3/19/90");
put_line(f,"-- Organization:  SAIC, Engineering Systems Group, San Diego, CA");
put_line(f,"-- ");
put_line(f,"-- ");
put_line(f,"-- Description:  User supplied.");
put_line(f,"-- ");
put_line(f,"--    NONE");
put_line(f,"-- ");
put_line(f,"-- ");
put_line(f,"-- Assumptions:");
put_line(f,"-- ");
put_line(f,"--    NONE");
put_line(f,"-- ");
put_line(f,"-- ");
put_line(f,"---------------------------------------------------------------------");
put_line(f," ");
put_line(f,"---------------------------------------------------------------------");
put_line(f,"-- Revision History:");
put_line(f,"-- Date      Author, Org.    Revision summary and validation
performed");
put_line(f,"-- ------------ --------------------------------------------------");
put_line(f,"--                                                                   ");
put_line(f,"-- ");
put_line(f,"---------------------------------------------------------------------");
put_line(f,"  ");
end SAIC_H;
```

The second example of a procedure that instantiated Ev_Gen is given below. In this example procedure, Gen_LH instantiates EV_Gen so that (1) file and program unit names created will begin with string "LH," (2) there will be code generated for 15 events, and (3) the procedure SAIC_H will be used to create the file header.

```
--------------------------------------------
-- Evada file generator for HF Long Haul model.
--------------------------------------------
with EV_Gen, ada_name, name, vms_acs, date, SAIC_H;
procedure GEN_LH is
  type EVENT_TYPE is
             (DSS_RCV, DSS_XMIT, DS_COUNTER, DS_RCV,
             DS_XMIT, GEN_BDCST, GEN_P2P, RESET, START_SLOT,
             TIMEOUT, TUNE_1, SOUND, TUNE_2, NET_CALL,
             LQA_XCHNG, NET_ACK, TUNE_3, START_ALE);
  procedure EV is new EV_Gen (Sim_NAME     => "LH",
                              EVENTS       => EVENT_TYPE,
                              File_Header => SAIC_H);
  begin
    EV("HF Long Haul");
end GEN_LH;
```

The header that is generated in example two is shown below.

```
-----------------------------------------------------------------
--          package spec LH
-----------------------------------------------------------------

-----------------------------------------------------------------
--          simulation: HF Long Haul
--          module:     Pkg Spec for Simulation Job Definitions
-----------------------------------------------------------------

-----------------------------------------------------------------
--
-- Author:         E. Beck
-- Date:           3/21/90
-- Organization:   SAIC, Engineering Systems Group, San Diego, CA
--
--
-- Description:  Lists constants connecting event types (jobs)
--               and numbers.
--               A few widely used data types are listed.
--
--
--
-- Assumptions:
--
--     NONE
--
--
-----------------------------------------------------------------

-----------------------------------------------------------------
-- Revision History:
-- Date       Author, Org.    Revision summary and validation performed
-- --------   -------------   ---------------------------------------
--
--
-----------------------------------------------------------------
```

3.2 COMPILING AND EXECUTING A PROCEDURE THAT INSTANTIATES Ev_Gen

The commands listed below, which are in a command file, are used to compile, link, and execute the procedure GEN_SOURCE_MV. (Refer to the first example above.)

```
! Set program library to [320032.EV.GEN].
$acs set lib [320032.EV.GEN]
! Compile instantiation procedure.
$ada/list Gen_Source_MV.ada
$acs link Gen_Source_MV              ! Gen_Source_MV is the name of
                                     ! procedure that instantiates
                                     ! package EV_Gen.
$run Gen_Source_MV
```

Upon execution, the following files are generated:

```
package spec MV
package body MV
procedure spec MV_Ep_Stop_Simulation
procedure body MV_Ep_Stop_Simulation
procedure spec MV_EP_MESSAGE_GENERATION
procedure body MV_EP_MESSAGE_GENERATION
procedure spec MV_EP_START_SLOT
procedure body MV_EP_START_SLOT
procedure spec MV_EP_TRANSMIT
procedure body MV_EP_TRANSMIT
procedure spec MV_EP_RECEIVE
procedure body MV_EP_RECEIVE
procedure spec MV_Ep_STOP_SERVICE
procedure body MV_Ep_STOP_SERVICE
procedure spec MV_DISPATCHER
procedure body MV_DISPATCHER
```

3.3 GAS STATION EXAMPLE

The use of EARS can best be illustrated by describing in detail a simulation example. The example given is the classic gas station model. This example requires the use of EARS packages, including EV, RANDOM, statistical packages, and linked list packages. The example illustrates the use of the packages and their subprograms. A more detailed explanation of these packages is found in subsequent chapters.

3.3.1 Statement of Problem

The problem is to develop a gas station simulation where the customers arrive randomly. Their interarrival rate is exponentially distributed. The customers queue up to be serviced by attendants, receive service, and then leave. The service times are randomly selected from an exponential distribution. The purpose of the simulation is to examine the effect of adding attendants and changing the mean interarrival time and the mean service time for simulations of varying duration.

3.3.2 Determining the Events

Before instantiating Ev_Gen, the user must determine the EARS events required for the simulation to be developed. These events are defined in the enumerated type Event_Type in the package that instantiates EV_Gen. An EARS event is an occurrence that takes place at some point in time. An EARS event may be scheduled to occur after a delta time or at a specific time.

For the gas station problem, only three things are scheduled: the arrival of a customer, the time required for service, and the end of the simulation. The names chosen for the event types are Customer_Arrival, Stop_Service, and Stop_Simulation. Remember, the Stop_Simulation procedure is generated automatically without defining it in Event_Type.

3.3.3 Writing the Procedure to Instantiate Package Ev_Gen

This procedure is given below. EV_Gen is only instantiated with the two required parameters, Sim_Name and Events. The string name for files and program units begins with "Hans." Only a Customer_Arrival and Stop_Service event are defined.

```
with Ev_Gen;
procedure Gen_Source_GS is
  type Event_Type is (Customer_Arrival, Stop_Service);
  procedure EV is new Ev_Gen (Sim_Name =>"GS",
                              EVENTS => Event_Type);
begin
  EV(User_Name => "Hans");
end Gen_Source_GS;
```

3.3.4 Compiling and Executing the Instantiation Procedure

A command file that contains the code required to compile, link, and execute the above procedure using VAX/VMS is given below.

```
! Set program library to [320032.EV.GEN].
$acs set lib [320032.EV.GEN]
! Create sublibrary [320032.EV.GEN.SUBLIB].
                                   ! Set to sublibrary
$ada/list Gen_Source_GS.ada        ! Compile instantiation procedure
$acs link Gen_Source_GS            ! Gen_Source_GS is the name of
                                   ! procedure that instantiates
                                   ! package EV_Gen.
$run Gen_Source_GS
! Upon successful execution of Gen_Source_GS the

$acs delete sublib [320032.EV.GEN.sublib]
```

3.3.5 Code That Is Generated Automatically

The ten program units listed in this section are generated at the completion of the above commands. An explanation of each program unit will follow its source code. Each line of code that is generated automatically will, for the sake of clarity, have "-- A" added to the right-hand side.

```
package spec GS
package body GS
procedure spec GS_Ep_STOP_SIMULATION
procedure body GS_Ep_STOP_SIMULATION
procedure spec GS_Ep_CUSTOMER_ARRIVAL
procedure body GS_Ep_CUSTOMER_ARRIVAL
procedure spec GS_Ep_STOP_SERVICE
procedure body GS_Ep_STOP_SERVICE
procedure spec GS_DISPATCHER
procedure body GS_DISPATCHER
```

3.3.5.1. *Automatically Generated Package Specifications*

The only package specification that is generated automatically is Gs. This procedure is given below. In this example, only the comments and two lines of Ada code are added by hand. The first line added by hand "withs in" package Gs_Dispatcher. The second instantiates the generic package Ev.Simulation with Ev.Max_Priority => 1 and Ev.Dispatcher => GS.Dispatcher. This means that for the gas station simulation all the events have the same scheduling priority and the dispatcher used will be GS.Dispatcher, which is generated automatically.

The next line of code assigns a value of 1 to Group. Group is a number that is associated with a specific simulation. Simulations that may potentially be integrated are assigned consecutive group numbers.

In the next three lines of code, the parameters Stop_Simulation, Customer_Arrival, and Stop_Service are defined as constants 0, 1, and 2, respectively. EV.Sim_Job is a unique integer that is associated with each type of event. It will be used in the dispatcher body.

Function Nu returns an event, given an input parameter of type Ev.Sim_Job. The event has two discriminants. Function Nu sets the first to the value of Job and the second to the value of Group. Section 4 will discuss type Ev.Event in more detail.

```
--| ------------------------------------------------------ -- A
--| Hans, MONDAY 12-MAR-1990 09:16.14                      -- A
-->>Pkg Spec for Simulation Job Definitions               -- A
--| ------------------------------------------------------ -- A
                                                           -- A
with Ev;                                                   -- A
with Gs_Dispatcher;
package Gs is                                              -- A
                                                           -- A
-- EV.Simulation is called with the following two parameters:
-- 1 represents the number of priorities in the simulation
-- GS_Dispatcher is the name of the dispatcher procedure that is generated
-- automatically by EV_Gen
  package Gs_Simulation is
    new Ev.Simulation (Max_Priority => 1, Dispatcher => Gs_Dispatcher);

  Group                                                    -- A
    : constant Ev.Sim_Group := 1;                          -- A

    -- Numbers assigned below are values of discriminant EV.Event_Rec, which
    -- are used in dispatcher.
```

continued

```
   Stop_Simulation                                        -- A
     : constant Ev.Sim_Job := 0;                          -- A
   Customer_Arrival                                       -- A
     : constant Ev.Sim_Job := 1;                          -- A

   Stop_Service                                           -- A
     : constant Ev.Sim_Job := 2;                          -- A

   function Nu                                            -- A
     ( Job : Ev.Sim_Job )                                 -- A
       return Ev.Event;                                   -- A
   --| Creates Events for this Event Group                -- A
 end Gs;                                                  -- A
```

3.3.5.2. *Automatically Generated Package Bodies*

The only package body generated automatically is Gs. It contains the function body Nu, which was described above. The package body Gs appears below.

```
--| ------------------------------------------------------ -- A
--| Hans, MONDAY 19-MAR-1990 14:56.48                      -- A
-->>Pkg body for Simulation Job Definitions                -- A
--| ------------------------------------------------------ -- A
 package body Gs is                                        -- A
  function Nu                                              -- A
    ( Job : Ev.Sim_Job )                                   -- A
      return Ev.Event is                                   -- A
    Evnt                                                   -- A
      : Ev.Event;                                          -- A

  begin -- Nu                                              -- A
    Evnt := new Ev.Event_Rec(Group, Job);                  -- A
    return Evnt;                                           -- A
  end Nu;                                                  -- A
end Gs;                                                    -- A
```

3.3.5.3. *Automatically Generated Procedure Specifications*

The gas station example has four procedure specifications that are generated automatically. Three procedures created are associated with event types. The fourth is the dispatcher. The event procedures created are GS_Ep_Customer_Arrival, GS_Ep_Stop_Service, and GS_Ep_Stop-Simulation. Each of the three event procedures has one input, Var, that is of type EV.Event. Procedure GS_Ep_Customer_Arrival is given below.

```
with Ev;                                                   -- A
procedure GS_Ep_Customer_Arrival (Var: Ev.Event);          -- A
--| ------------------------------------------------------ -- A
--| Hans, MONDAY 12-MAR-1990 09:16.16                      -- A
-->>Event Procedure for Event Type 0                       -- A
--| ------------------------------------------------------ -- A
```

Procedure GS_Ep_Stop_Service is given below.

```
with Ev;                                                -- A
procedure GS_Ep_Stop_Service (E: Ev.Event);             -- A
--| --------------------------------------------------- -- A
--| Hans, MONDAY 12-MAR-1990 09:16.17                   -- A
-->>Event Procedure for Event Type 1                    -- A
--| --------------------------------------------------- -- A
```

Procedure GS_Ep_Stop_Simulation is given below.

```
with Ev;                                                    -- A
procedure GS_Ep_Stop_Simulation (Evnt : Ev.Event);          -- A
--| ------------------------------------------------------- -- A
--| Hans, MONDAY 12-MAR-1990 09:16.15                       -- A
-->>Event Procedure to Stop the Simulation at a Specific time -- A
--| ------------------------------------------------------- -- A
```

The fourth specification, for procedure GS_Dispatcher, also has one input, Evnt, that is of type EV.Event. This procedure is given below.

```
--| --------------------------------------------------- -- A
--| Hans, MONDAY 19-MAR-1990 14:02.07                   -- A
-->>Event Procedure Dispatcher                          -- A
--| --------------------------------------------------- -- A
                                                        -- A
with Ev;                                                -- A
procedure GS_Dispatcher (Evnt: Ev.Event);               -- A
```

3.3.5.4. *Automatically Generated Procedure Bodies*

The bodies for the four procedures, GS_Ep_CUSTOMER_ARRIVAL, GS_Ep_STOP_SERVICE, GS_Ep_Stop_Simulation, and GS_Dispatcher, are also generated automatically.

First, the event procedures will be discussed. At the top of each procedure the string, Name, is defined to contain the name of the procedure. Name is printed in the exception handler when an error occurs.

Each procedure body will now be discussed individually.

1. Procedure Gs_Ep_Customer_Arrival

The procedure GS_Ep_Customer_Arrival is given below. Variables Customer_Service_Time, Next_Arrival, and Gs_Xp.Mean_Service_Time are of type Sim.Real. Sim.Real is the EARS definition of type "float," with 9 decimal digits of accuracy. Following the "begin" statement, function Random.Exponential is called with two inputs, the random number stream selected, which is 1, and the mean service time, which is the value of

GS_Xp.Mean_Service_Time. A stream is a source of random numbers. There is a random number seed associated with each stream. GS_Xp.Mean_Service_Time is a user input to the package specification Gs_Xp. Procedure Random.Exponential returns a random number from an exponential distribution whose mean is the value of GS_Xp.Mean_Service_Time.

The procedure Tf.Assign is used to maintain tallied floating point statistics on Service_Time. TF.Assign assigns the value of Customer_Service_Time to the statistical variable GS_Xp.St.

Next, there is a "for loop" in which each attendant is checked to see if he is busy. When he is not busy, he is set to busy. The start work time for the attendant is set equal to the value of the simulation clock, Ev.Clock. Initially it is zero. The procedure Ev.Delta_Schedule has two inputs: Customer_Service_Time and an attendant event. When all the attendants are busy, then the customer is put into the customer queue, GS_Xp.Customer_Queue. This is accomplished by the call to Rl_Ai.Append. Rl_Ai is a package that is an instantiation of linked list package Lnk_Ai with type Sim.Real. Lnk_Ai appends the member Customer_Service_Time to the end-of-link list Gs_Xp.Customer_Queue. Package Lnk_Ai is an EARS linked list package that automatically maintains time-dependent statistics on queue length. In this case, it is the Gs_Xp.Customer_Queue.

Next, the exponentially distributed random variable Next_Arrival time is selected by calling procedure Random.Exponential with the first stream and a mean of Gs_Xp.Mean_Interarrival_Time. Finally, Ev.Delta_Schedule is called to schedule the next arrival. The next arrival is scheduled to occur at time EV.Clock plus Next_Arrival time.

```
with Sim;                                                  -- A
with Rl_Ai, Random, Gs_Xp, Gs_Ev, Tf;
procedure Gs_Ep_Customer_Arrival                           -- A
  ( Var : Ev.Event ) is                                    -- A
  Name                                                     -- A
    : constant STRING := "GS_Ep_CUSTOMER_ARRIVAL";         -- A

  Customer_Service_Time
    : Sim.Real;

  Next_Arrival
    : Sim.Real;

begin -- Gs_Ep_Customer_Arrival                            -- A
--| Determine Customer Service time.
  Customer_Service_Time := Random.Exponential(1, Gs_Xp.Mean_Service_Time);
  Tf.Assign(Gs_Xp.St, Customer_Service_Time);
  --| Try to find an idle Attendant
  for This in Gs_Xp.Attendants loop
    if not Gs_Xp.Attendant(This).Busy then
      Gs_Xp.Attendant(This).Busy := TRUE;
      Gs_Xp.Attendant(This).Start_Work := Ev.Clock;
      -- For EV.Delta_Schedule if no event and priority are passed
      -- as parameters then default is current event and priority.
      Ev.Delta_Schedule(
        Time => Customer_Service_Time,
        Evnt => Gs_Ev.Stop_Service(This));
      exit;
    elsif This = Gs_Xp.Attendants'Last then
    --| All Attendants are busy, so customer goes in queue
      Rl_Ai.Append(Gs_Xp.Customer_Queue, Customer_Service_Time);
```

continued

```
    end if;
  end loop;
  Next_Arrival := Random.Exponential(1, Gs_Xp.Mean_Interarrival_Time);
  Ev.Delta_Schedule(Time => Next_Arrival, Evnt => Var);
  -- Next line is deleted when do simulation
  -- raise Sim.Not_Implemented;                                   -- A
exception -- Gs_Ep_Customer_Arrival                               -- A
  when others =>                                                  -- A
    Ev.Err_IO("Crashed in " & Name);                              -- A
    raise;                                                        -- A
end Gs_Ep_Customer_Arrival;                                       -- A
```

2. Procedure Gs_Ep_Stop_Service

The procedure GS_Ep_Stop_Service is shown below. The first statement following "begin" updates the time worked for an attendant. The value of E.Var(0), by convention, is used as an identification number. In this example, it represents the attendant ID. (For communications network modeling it might represents Node ID.)

The next statement updates the number of customers served by the attendant. Then the function Rl_Ai.Nl checks to see if the number of items in the customer queue is greater than zero. When it is, then the Start_Work variable for the attendant is set equal to current simulation time, Ev.Clock. Procedure Rl_Ai.Pop pulls Customer_Service_Time off the front of the Customer_Queue. Next, the procedure Ev.Delta_Schedule schedules the time for service completion. When the number of items in the customer queue is zero, the attendant is set to not busy.

```
with Ev;                                                          -- A
with Gs_Xp, Rl_Ai;
with Sim;                                                         -- A
use Sim;
procedure Gs_Ep_Stop_Service                                      -- A
  ( E : Ev.Event ) is                                             -- A
  Name                                                            -- A
    : constant STRING := "GS_Ep_STOP_SERVICE";                    -- A

  Customer_Service_Time
    : Sim.Real;

  Found
    : BOOLEAN;

begin -- Gs_Ep_Stop_Service                                       -- A
--| Update Stats for Attendant
-- E.Var(0) contains the ID of attendant
  Gs_Xp.Attendant(E.Var(0)).Time_Worked
        := Gs_Xp.Attendant(E.Var(0)).Time_Worked
        + Ev.Clock - Gs_Xp.Attendant(E.Var(0)).Start_Work;
  Gs_Xp.Attendant(E.Var(0)).Customers_Served
        := Gs_Xp.Attendant(E.Var(0)).Customers_Served + 1;
  --| Check Customer Queue
  -- Function NL returns the number of items in the list
  if Rl_Ai.Nl(Gs_Xp.Customer_Queue) > 0 then
  --| Start Customer Service.
```

continued

```
    Gs_Xp.Attendant(E.Var(0)).Start_Work := Ev.Clock;
    Rl_Ai.Pop(Gs_Xp.Customer_Queue, Customer_Service_Time, Found);
    --| Schedule Service Completion
    Ev.Delta_Schedule(Time => Customer_Service_Time, Evnt => E);
  else
  --| No customers, so not busy
    Gs_Xp.Attendant(E.Var(0)).Busy := False;
  end if;
  -- Next line is removed for simulation
  -- raise Sim.Not_Implemented;                              -- A
exception                                                    -- A
  when others =>                                             -- A
    Ev.Err_IO("crashed in "&name);                           -- A
    raise;                                                   -- A
end Gs_Ep_Stop_Service;                                      -- A
```

3. Procedure GS_Ep_Stop_Simulation

This procedure, shown below, sets the boolean value Ev.Simulating to "False," which causes the simulation to end.

```
procedure Gs_Ep_Stop_Simulation                             -- A
  ( Evnt : Ev.Event ) is                                    -- A
  Name                                                      -- A
    : constant STRING := "Gs_Ep_Stop_Simulation";           -- A
begin -- Gs_Ep_Stop_Simulation                              -- A
  Ev.Simulating := False;                                   -- A
exception -- Gs_Ep_Stop_Simulation                          -- A
  when others =>                                            -- A
    Ev.Err_IO("Crashed in " & Name);                        -- A
    raise;                                                  -- A
end Gs_Ep_Stop_Simulation;                                  -- A
```

4. Procedure GS_Dispatcher

This procedure, given below, is called by procedure Ev.Start with an input parameter that is an EV.Event. GS_Dispatcher looks at the value of the discriminant Evnt.Job and calls the appropriate event procedure GS_Ep_Customer_Arrival, GS_Ep_Stop_Service, or GS_Ep_Stop_Simulation. An exception is raised and an error message is printed for invalid values of Evnt.Job.

```
with GS;                                                    -- A
use GS;                                                     -- A
with GS_Ep_Customer_Arrival;                                -- A
with GS_Ep_Stop_Service;                                    -- A
with GS_Ep_Stop_Simulation;                                 -- A
procedure GS_Dispatcher (Evnt: Ev.Event) is                 -- A
  No_Such_Job : exception;                                  -- A
```

continued

```
begin                                                           -- A
  case Evnt.Job is                                              -- A
    when Customer_Arrival => GS_Ep_Customer_Arrival (Evnt);     -- A
    when Stop_Service => GS_Ep_Stop_Service (Evnt);             -- A
    when Stop_Simulation => GS_Ep_Stop_Simulation (Evnt);       -- A
    when others => raise No_Such_Job;                           -- A
  end case;                                                     -- A
exception                                                       -- A
  when No_Such_Job =>                                           -- A
    Ev.Err_IO                                                   -- A
    ("No_Such_Job as"&EV.Sim_Job'image(Evnt.Job)&" in           -- A
GS_Dispatcher");                                                -- A
    raise;                                                      -- A
  when others =>                                                -- A
    raise;                                                      -- A
end GS_Dispatcher;                                              -- A
```

3.3.6 New Program Units Added by Hand

1. Package Gs_Xp Specification and Body

The package specification Gs_Xp, given below, reads in simulation input data at "elaboration" and defines an array of records containing attendant information.

Function Int_Param is called to get the values for Number_Of_Attendants, Mean_Interarrival_Time, Mean_Service_Time, and Run_Time. Customer_Queue is a Lnk_Ai linked list whose item is of type Sim.Real. St is a statistical variable for tallied fixed point statistics. Attendant is an array of records containing attendant information. The procedure Report is called from procedure Gs_Station to print out results of the simulation.

```
with Sim, Int_Param, Real_Param, Rl_Ai, Tf;

package Gs_Xp is

  Number_Of_Attendants
    : constant POSITIVE
      := Int_Param("ATTENDANTS:");

  subtype ATTENDANTS is
    POSITIVE range 1 .. Number_Of_Attendants;

  Mean_Interarrival_Time
    : constant Sim.Real := Real_Param("MEAN INTERARRIVAL TIME (min):");

  Mean_Service_Time
    : constant Sim.Real := Real_Param("MEAN SERVICE TIME (min):");

  Run_Time
    : Sim.Real
      := Sim."*"(60.0, Real_Param("RUN_TIME (hrs):"));

  Customer_Queue
```

continued

```
    : Rl_Al.List;
  St
    : Tf.Variable;

  type ATTENDANT_REC is
    record
      Busy : BOOLEAN;
      Start_Work : Sim.Real;
      Time_Worked : Sim.Real := 0.0;
      Customers_Served : NATURAL := 0;
    end record;

  Attendant
    : array (ATTENDANTS)
      of ATTENDANT_REC;

  procedure Report;

end Gs_Xp;
```

The package body, given below, defines and initializes the Customer_Queue. The procedure Report prints out the simulation input parameters and the results. The results include the simulation time and customer queue statistics, which are printed from procedure Ai.Report and service time statistics, which are printed from procedure Tf.Report. Next, the percentage of time worked and the number of customers served by each attendant are printed.

```
with Integer_Text_IO, Text_IO, Ai;
with Ev;
use Integer_Text_IO, Text_IO;

package body Gs_Xp is
  CUSTOMER_QUEUE                -- INIT is executed upon elaboration

    : Rl_Ai.List := Rl_Ai.Init;

  procedure Report is
    use Sim, Sim.Io;
    Delim
      : constant STRING :=
"*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*";
    Title
      : constant STRING :=
        "                GAS STATION SIMULATION REPORT";

    Indent
      : constant Positive_Count := 4;

    Col
      : constant Positive_Count := 20;

    Ifw
      : constant POSITIVE := 10;

    Dfw
      : constant POSITIVE := 3;

    Rfw
      : constant POSITIVE := Ifw - Dfw - 1;

                                                  continued
```

```
   begin -- Report
     Set_Col(Indent);
     Put_Line(Delim);
     Set_Col(Indent);
     Put_Line(Title);
     Set_Col(Indent);
     Put_Line(Delim);
     Set_Col(Indent);
     Put_Line("INPUT PARAMETERS");
     Set_Col(Indent);
     Put("Attendants");
     Set_Col(Col);
     Put(Number_Of_Attendants, Ifw);
     New_Line;
     Set_Col(Indent);
     Put("Mean Interarrival Time");
     Set_Col(Col);
     Put(Mean_Interarrival_Time, Rfw, Dfw, 0);
     New_Line;
     Set_Col(Indent);
     Put("Mean Service Time");
     Set_Col(Col);
     Put(Mean_Service_Time, Rfw, Dfw, 0);
     New_Line;
     Set_Col(Indent);
     Put("Run Time");
     Set_Col(Col);
     Put(Run_Time, Rfw, Dfw, 0);
     New_Line;
     Set_Col(Indent);
     Put_Line(Delim);
     Set_Col(Indent);
     Put_Line("RESULTS");
     Set_Col(Indent);
     Set_Col(Indent);
     Put("SIMULATION TIME");
     Set_Col(Col);
     Put(Ev.Clock, Rfw, Dfw, 0);
     New_Line;
     Set_Col(Indent);
     Put_Line("CUSTOMER QUEUE");
     Set_Col(Indent);
     Ai.Report(Rl_Ai.Var(Customer_Queue));
     Set_Col(Indent);
     Put_Line("SERVICE TIME");
     Set_Col(Indent);
     Tf.Report(St);
     Set_Col(Indent);
     Put_Line(Delim);
     for This in Attendants loop
       Set_Col(Indent);
       Put_Line("Attendant" & INTEGER'Image(This) & " ");
       Set_Col(Indent);
       Put("Time Worked");
       Set_Col(Col);
       Put((Attendant(This).Time_Worked / Ev.Clock) * 100.0, Rfw, 2, 0);
       Put("%");
       New_Line;
       Set_Col(Indent);
       Put("Customers Served");
       Set_Col(Col);
       Put(Attendant(This).Customers_Served, Ifw);
       New_Line;
     end loop;
     Put_Line(Delim);
     New_Line;
   end Report;
 end Gs_Xp;
```

2. Package Gs_Ev Specification and Body

The package specification Gs_Ev, given below, defines Stop_Service as an array of events. Customer_Arrival and Stop_Simulation are defined as events.

```
with Gs_Xp, Ev;

package Gs_Ev is

  Stop_Service
    : array (Gs_Xp.Attendants)
      of Ev.Event;

  Customer_Arrival
    : Ev.Event;

  Stop_Simulation
    : Ev.Event;

end Gs_Ev;
```

The package body, below, loops on the number of attendants. Within the loop, STOP_SERVICE, indexed for each attendant, is assigned an event. The next statement uses the procedure EV.Set_Priority to set the scheduling priority of the event record to 1. In this simulation, all events have a priority of 1.

The next statement sets the size of the array of integers pointed to by Stop_Service.VAR to 1. The following statement initializes the value of this array to index ID for the attendant. In this simulation, only one ID number is needed: the attendant ID. In other simulations, larger arrays may be needed to store other ID numbers.

Following the loop, events are created for Customer_Arrival and Stop_Simulation. Then the scheduling priorities for each are set to 1.

```
with GS;
package body GS_EV is
begin
  for ID in GS_XP.ATTENDANTS loop
    STOP_SERVICE(ID) := GS.NU(GS.STOP_SERVICE); -- Creates events
                                                 -- for Stop_Service.
    EV.set_priority (STOP_SERVICE(ID),1 ); -- Assigns priority
                          -- of 1 to stop service.
    Stop_Service(ID).VAR := new EV.Array_of_Integer(0..0); -- Set
                          -- range of VAR.
    Stop_Service(ID).Var(0) := ID; -- Assign attendant ID to index
                     -- ID.
  end loop;                              -- Attendant ID number is the
                  -- only ID have in gas station
                  -- simulation.
    Customer_Arrival := GS.NU(GS.Customer_Arrival); -- Creates
                -- events for Customer_Arrival.

                                                          continued
    Stop_Simulation := GS.NU(GS.Stop_Simulation);  -- Creates
         -- events for Stop_Simulation.
    EV.set_priority (Customer_Arrival,1);    -- Assigns priority of
            -- 1 to customer_arrival.
    EV.set_priority (Stop_simulation,1);     -- Assigns priority of
            -- 1 to stop_simulation

end GS_EV;
```

3. Procedure GS_Station

The driver procedure, GS_Station, defines Customer_Generator and Stop_Sim to be events, while Arrival is of type Sim.Real. This procedure is shown below. The procedure begins by randomly selecting an exponentially distributed arrival time when using the first stream of random numbers and a user-specified mean interarrival time. The next two lines make Customer_Generator an event whose discriminants, Job and Group, both have a value of 1. The next two statements do the same thing for Stop_Sim.

Then a customer arrival is scheduled to occur after the delta time Arrival, and simulation is scheduled to stop after the delta time Gs_Xp.Run_Time. The next statement starts the simulation. Gs.Gs_Simulation.Start is a call to the generic package Ev.Simulation, which contains procedure Start. Finally, procedure Gs_Xp.Report prints the simulation inputs and results.

```
with Ev;
with Sim, Random, Gs, Gs_Ev, Gs_Xp;

procedure Gs_Station is
  Customer_Generator
    : Ev.Event;

  Stop_Sim
    : Ev.Event;

  Arrival
    : Sim.Real;

begin -- Gs_Station
  Arrival := Random.Exponential(1, Gs_Xp.Mean_Interarrival_Time);

  Customer_Generator := Gs.Nu(Gs.Customer_Arrival);
  Ev.Set_Priority(Customer_Generator, 1);
  Stop_Sim := Gs.Nu(Gs.Stop_Simulation);
  Ev.Set_Priority(Stop_Sim, 1);
  Ev.Delta_Schedule(Time => Arrival, Evnt => Customer_Generator);
  Ev.Delta_Schedule(Time => Gs_Xp.Run_Time, Evnt => Stop_Sim);
  Gs.Gs_Simulation.Start;
  Gs_Xp.Report;
end Gs_Station;
```

3.3.7 Compilation Order

The compilation order for the gas station problem is given below.

```
ada/list gs_dispatcher_.ada
ada/list gs_.ada
ada/list gs.ada
ada/list gs_xp_.ada
ada/list gs_xp.ada
ada/list gs_ev_.ada
ada/list gs_ev.ada
ada/list gsep_stop_simulation_.ada
ada/list gsep_stop_simulation.ada
ada/list gs_ep_customer_arrival_.ada
ada/list gs_ep_customer_arrival.ada
ada/list gs_ep_stop_service_.ada
ada/list gs_dispatcher.ada
ada/list gs_ep_stop_service.ada
ada/list gs_station.ada
```

4.0 USING PACKAGE EV

This section, contains a description of package EV. All EARS simulations use data structures and subprograms that are defined in package EV. First, the package specification will be discussed and shown. Then specific examples will be given that illustrate a use of the more complex subprograms.

4.1 EV PACKAGE SPECIFICATION

The user should refer to the EV package specification given below when reading this section. This specification has been heavily commented. Many details not discussed in the text of this section are provided in the comments. The most relevant aspects of this package will be described in the sections that follow.

```
with Sim, Text_IO, Hex;
use Sim, Text_IO;
package Ev is

  -- The simulation clock is initialized to 0.
  Clock
    : Real := 0.0;

  -- Stop_Immediately is an exception that may
  -- be used by the modeler. It is not used in
  -- package EV. Stop_Immediately and Simulating
  -- provide a capability for graceful simulation
  -- termination.
  Stop_Immediately
    : exception;

  Simulating
    : BOOLEAN := FALSE;

  -- Immediately represents the "earliest possible" time.
  Immediately
    : constant Real := Real'First;

  -- No_Priority_Change is used in procedures
  -- EV.Delta_Schedule and EV.Abs_Schedule to indicate
  -- that the priority does not change.
  No_Priority_Change
    : constant INTEGER := 0;

  -- The lowest usable priority is 1.
  Lowest_Priority
    : constant INTEGER := INTEGER'Succ(No_Priority_Change);

  -- The highest priority that a user can specify is 31.
  Highest_Priority
    : constant INTEGER := 31;

  -- Defines range of scheduling priorities that
  -- are used internally in EVADA.
  subtype EVENT_PRIORITY_RANGE is
    NATURAL range No_Priority_Change .. Highest_Priority;

  -- Defines range of scheduling priorities
  -- available to the user.
  subtype EVENT_PRIORITY is
    NATURAL range Lowest_Priority .. Highest_Priority;
```

continued

```
-- Defines type for array of integers that is used for
-- IDs in Event_Rec.
type ARRAY_OF_INTEGER is
  array (NATURAL range <>)
    of INTEGER;

-- EVENT_PARAM points to array of integers where ID
-- numbers are stored.
type EVENT_PARAM is
  access ARRAY_OF_INTEGER;

-- Evada is used to represent events that are internal
-- to EVADA.
Evada
  : constant := 0;

-- Last_Group is the largest Group number that a user
-- may use.
Last_Group
  : constant := 31;

-- GROUP_TYPE is range of Group numbers used internally
-- in EVADA.
type GROUP_TYPE is
  new INTEGER range Evada .. Last_Group;

-- First_Usable_Group represents the first Group number
-- that a user may use.
First_Usable_Group
  : constant GROUP_TYPE := GROUP_TYPE'Succ(Evada);

-- Sim_Group is the range of Group numbers that the user
-- may use.
subtype SIM_GROUP is
  GROUP_TYPE range First_Usable_Group .. GROUP_TYPE'Last;

-- First_Job and Last_Job are the minimum and maximum
-- job numbers that a user may use.
First_Job
  : constant := 0;

Last_Job
  : constant := 127;

-- SIM_JOB is a type that spans the range job numbers.
type SIM_JOB is
  new INTEGER range First_Job .. Last_Job;

-- Control contains event control information which the
-- user need not know.
type CONTROL is
  limited private;

-- EVENT_REC is the EVADA event record.
-- The discriminant Group is a number that allows the
-- correct dispatcher to be selected for an event. It is
-- only used for integrating simulations.
-- The discriminant Job is a number that corresponds to
-- an Event type that the user defines in the procedure
-- that instantiates EV_Gen. Job is needed when simulations
-- are integrated.
-- Var points to the array of integers that is used for
-- storing ID numbers.
```

continued

```
-- Ctrl is event control information that the user need
-- not know about.
type EVENT_REC
  ( Group : GROUP_TYPE;
    Job : SIM_JOB ) is
  record
    Var : EVENT_PARAM;
    Ctrl : CONTROL;
  end record;

  type EVENT is
  access EVENT_REC;

-- Ent is the currently executing event.
-- It can be used by procedures EV.Delta_Schedule and
-- EV.Abs_Schedule.
Ent
  : EVENT;

-- Procedure Delta_Schedule schedules an event to become active
-- at a simulated time of current time plus "Time". Parameter PRI
-- allows the user to change the priority.
procedure Delta_Schedule
  ( Time : Sim.Real;
    Evnt : EVENT := Ent;
    Pri : EVENT_PRIORITY_RANGE := No_Priority_Change );

-- Procedure Abs_Schedule schedules an event to become active
-- at an absolute simulated time of "time".
procedure Abs_Schedule
  ( Time : Sim.Real;
    Evnt : EVENT := Ent;
    Pri : EVENT_PRIORITY_RANGE := No_Priority_Change );

-- Procedure Set_Priority changes the priority in Evnt to
-- Pri.
procedure Set_Priority
  ( Evnt : EVENT;
    Pri : EVENT_PRIORITY_RANGE );

-- Procedure Remove removes an event from the ready queue.
-- Dispose = true means the event record is destroyed.
procedure Remove
  ( Evnt : in out EVENT;
    Dispose : BOOLEAN := FALSE );

-- Function Activation_Time returns the activation time
-- that an event is scheduled to occur.
function Activation_Time
  ( Evnt : EVENT )
    return Real;

-- Function Priority returns the priority of an event.
function Priority
  ( Evnt : EVENT )
    return EVENT_PRIORITY;

-- Package Simulation is called with a maximum number of
-- priorities that the user needs and with the dispatcher
-- procedure that is generated automatically for the
-- users simulation.
generic

  Max_Priority
    : EVENT_PRIORITY; -- The highest priority required by
                      -- simulation.
```

continued

```
  with procedure Dispatcher -- The dispatcher procedure
    ( Evnt : EVENT );   -- generated by the user instantiation
                        -- and execution of EV_Gen.
package Simulation is

  procedure Start;      -- Starts the simulation.

end Simulation;

Fore
  : NATURAL := 2;

Aft
  : NATURAL := 6;

Exp
  : NATURAL := 0;

-- Function Str, Time, and Id are used for
-- debugging purposes.

-- Function Str returns the hex representation of an
-- event. It is used for debugging purposes.
function Str is
  new Hex (EVENT);

-- Function Time converts time from type Sim.Real
-- to string.
function Time
  ( T : Sim.Real )
    return STRING;

-- Function Id converts Group and Job to
-- string "[Group,Job]".
function Id
    return STRING;
--| "[Group,Job]"

-- Procedures Switch_SIO, SIO, Switch_FIO, Create,
-- Close, Fio, and Err_IO below are also intended
-- for debugging purposes. They may be called with
-- or without the debugger. A modeler may wish
-- to use the debugger to change the values
-- of the next four flags by using the "deposit"
-- feature of the VAX debugger.

-- Sio_On = TRUE means output to monitor.
-- Sio_On is examined in procedure Sio.
Sio_On
  : BOOLEAN := FALSE;

-- Sio_Delta = TRUE means scheduling occurs in
-- delta time.
-- Sio_Delta = false means scheduling is in
-- absolute time.
Sio_Delta
 : BOOLEAN := TRUE;

-- Sio_Time is scheduling time for procedure Switch_Sio.
Sio_Time
  : Sim.Real;

-- The scheduling priority for Switch_Sio is set to 1.
Sio_Pri
  : EVENT_PRIORITY
    := EVENT_PRIORITY'First;

-- Procedure Switch_Sio specifies the time at which
```
continued

```
     procedure SIO is effective or ineffective.
procedure Switch_Sio
    ( Time : Sim.Real := Sio_Time;
      Pri : EVENT_PRIORITY := Sio_Pri;
      Delta_Sio : BOOLEAN := Sio_Delta;
      On : BOOLEAN := Sio_On );

  -- When Sio_On is true procedure Sio print out
  -- Clock, Group, Job, and string to monitor.
  procedure Sio
    ( Str : STRING;
      Eol : BOOLEAN := TRUE );

  -- Fio_On = TRUE means output to file.
  Fio_On
    : BOOLEAN := FALSE;

  -- Fio_Delta = TRUE means scheduling occurs in
  -- delta time.
  -- Fio_Delta = FALSE means scheduling  is in
  -- absolute time.
  Fio_Delta
   : BOOLEAN := TRUE;

  -- Fio_Time is sceduling time for procedure Switch_FIO.
  Fio_Time
    : Sim.Real;

  -- Scheduling priority for Switch_FIO is set to 1.
  Fio_Pri
    : EVENT_PRIORITY
      := EVENT_PRIORITY'First;

  -- Procedure Switch_Fio works like Switch_Sio,
  -- except that output is to a file.
  -- (The file is named in a string that is input
  -- to procedure Create below.)
  procedure Switch_Fio
    ( Time : Sim.Real := Fio_Time;
      Pri : EVENT_PRIORITY := Fio_Pri;
      Delta_FIO : BOOLEAN := Fio_Delta;
      On : BOOLEAN := Fio_On );

  F
    : File_Type;

  -- Procedure Create uses Text_IO.Create
  -- to create F with file_name = Fn,
  -- mode out_file.
  procedure Create
    ( Fn : STRING );

  -- Procedure Close uses Text_IO.Close
  -- to close F.
  procedure Close;

  -- When Fio_On is true procedure Fio writes
  -- out Clock, Group, and string to Fn, which is
  -- is input to procedure Create.
  procedure Fio
    ( Str : STRING;
      Eol : BOOLEAN := TRUE;
      File : File_Type := F );
```

continued

```
    -- Procedure Err_IO writes Date and Time,
    -- Str (Ent), ID, Clock, and Message.
    -- The input, Msg, is any user defined string,
    -- such as "Crashing in procedure xxxx".
    procedure Err_IO
      ( Msg : STRING := "" );

    -- Beyond this point are implementation details
    -- that are not of concern to the user.

  -- Private portion of specification deleted in
  -- this documentation.

    type CONTROL is
      record
        Activation_Time : Real;
        Priority : EVENT_PRIORITY;
        Queued : BOOLEAN := FALSE;
        Next : Event := null;
        Prev : Event := null;
      end record;

    Bk_Time  -- See Delta_Bk and Abs_Bk below.
      : Sim.Real;

    Bk_Pri   -- See Delta_Bk and Abs_Bk below.
      : Event_Priority := 1;

    -- The next three procedures are used with a debugger.

    -- Procedure Breakpoint is a null procedure that is
    -- intended to be used to get to command mode in a
    -- symbolic debugger.
    procedure Breakpoint;

    -- Procedure Delta_Bk calls EV.Delta_Schedule using
    -- the parameters BK_Time, Break_Ev, and Bk_Pri
    -- which are defined above. The values for these
    -- parameters are defined using the debugger.
    procedure Delta_Bk;
    -- Considers Bk_Time a Delta time and schedules procedure
    -- "Breakpoint" to occur when Clock = Now + Bk_Time, at
    -- priority = Bk_Pri.

    -- Procedure Abs_Bk calls EV.Abs_Schedule and is
    -- used in the same manner as Delta_Bk.
    procedure Abs_Bk;

  end Ev;
```

4.1.1 EVENT_REC

The EARS event record, EVENT_REC, contains scheduling and other information that is associated with an event. EVENT_REC contains the discriminants Group and Job. The intent of the discriminants is to allow different simulations to be integrated. Group is a number that is associated with a simulation. Simulations that may potentially be integrated should be assigned different Group numbers. A user may assign a Group number from 1 to 127. The Job number is associated with the event types that are defined in the EV_Gen instantiation procedure. Job numbers range from 1 to 31. Section 3.3.5 illustrates the assignment of Job and Group.

The parameter Var points to an array of integers that contains identification numbers. For example, in the gas station simulation, there was an array of one number that contained an attendant ID. Whereas, for a communication network simulation, the array may contain node IDs. The Ctrl parameter contains information, such as scheduling activation time, that the user need not be concerned about.

4.1.2 Event Processing Subprograms

The subprograms that relate to the processing of events are listed below. The package specification provides the details needed for calling them. Procedures Delta_Schedule and Abs_Schedule allow events to be scheduled to occur in a delta time and at an absolute time. The scheduling priority may be set in the calls to Procedure Delta_Schedule, Abs_Schedule, and Set_Priority. Procedure Remove is used to remove an event from the event queue and to optionally destroy it. Function Activation_Time and Priority return the event activation time and priority.

```
procedure Delta_Schedule
procedure Abs_Schedule
procedure Set_Priority
procedure Remove
function Activation_Time
function Priority
```

In EARS, priorities are used to break ties when more than one event is scheduled at the same time. The event with the highest priority is scheduled first.

4.1.3 Package Simulation

The user must instantiate the generic package Simulation with Max_Priority, the maximum number of scheduling priorities needed in a simulation (permissible values are 1 to 31), and the dispatcher procedure that was generated automatically. The example below illustrates how to instantiate this package with 18 scheduling priorities and the dispatcher procedure LH_Dispatcher.

```
package LH_Simulation is
  new Ev.Simulation (Max_Priority => 18,
                     Dispatcher => LH_Dispatcher);
```

4.1.4 General Debugging Subprograms

The subprograms listed below are intended for debugging purposes either with or without a debugger. The comments in the EV package specification explain their use.

```
function Str
function Time
function Id
procedure Switch_Sio
procedure Sio
procedure Switch_Fio
procedure Create
procedure Close
procedure Fio
procedure Err_IO
```

1. Switch_Sio, Sio Example

The example below illustrates the use of Switch_Sio and Sio. These procedures are used to examine a sequence of events. The call to Switch_SIO schedules an event to occur in a delta simulation time 20 with a priority of 1. The On flag is set to "true" when this event occurs. This tells procedure Sio to print.

```
EV.Switch_SIO(Time => 20, Pri => 1, On => true);
```

The Sio call, below, is placed in an event procedure whose timing is being examined. The statement prints the user defined string followed by Clock, Group, and Job. The printing only happens when the On flag is set to "true."

```
EV.Sio(Str => "Transmit for Node xx");
```

2. Switch_Fio, Fio Example

The following set of statements illustrates the use of Create, Switch_Fio, Fio, Create, and Close. This set of statements is provided to examine a sequence of events and to print relevant information to a file. First, the output file must be created with the statement

```
EV.Create(Fn => "LH_Test_File");
```

The call to Switch_FIO schedules an event to occur in absolute simulation time 3 with a priority of 2. The On flag is set to true when this event occurs. This tells Fio to print.

```
EV.Switch_Fio(Time => 3, Pri => 2, Delta_Fio => false,
              On => true);
```

The statement below is placed in the relevant event procedure. The user-defined string, followed by Clock, Group, and Job, is printed to the output file. Printing only begins when the Switch_Fio On flag is set to "true."

```
EV.Fio(Str => "Transmit for Node yy");
```

The next statement causes the printing to be disabled at absolute simulation time 1.

```
EV.Switch_FIO(Time => 1, Delta_Fio => false, On => false);
```

Finally the file is closed with

```
EV.Close;
```

4.1.5 Debugger Subprograms

The procedures and declarations listed below are designed for use with a debugger.

```
Bk_Time  : Sim.Real;
Bk_Pri   : Event_Priority;
procedure Breakpoint;
procedure Delta_Bk;
procedure Abs_Bk;
Break_Ev  : Event;
Break_Job : Sim_Job;
```

These procedures allow a user to reduce execution time when running a simulation with the debugger. Sometimes a simulation may run for 8 or 10 hours before a breakpoint is reached. Using these routines can substantially reduce this time.

Their use can best be illustrated with an example. Consider the following scenario: Assume that something unexpected occurred at 150.0 seconds into the simulation after procedure X has executed more than 1000 times.

Without the aforementioned procedures, the user typically decides to track down the problem by setting a breakpoint to occur at procedure X at time 150.0. This means that every time procedure X is called, the debugger stops and the value of the simulation clock is checked. Doing this increases the execution time by orders of magnitude over the time needed to reach that point in the simulation when not using the debugger.

The user takes advantage of the breakpoint capability (in VAX/VMS) by executing the following commands inside the debugger:

```
DBG> set module Ev
DBG> deposit Ev.Bk_Time = 149.999
        -- Set time to break
DBG> set break Ev.Breakpoint
        -- Tell debugger to break here
DBG> call Abs_Bk
        -- Schedules Ev.Breakpoint at
        -- absolute time = 149.999
DBG> go
```

The debugger does no checking at all until procedure Ev.Breakpoint occurs at a simulation time of 149.999. When it is reached, the debug prompt is displayed and the user then enters the following:

```
DBG> set break X
DBG> go
```

Again, the debugger does no checking until procedure X is called. The debugger then stops and gives the debug prompt. Execution speed under this scenario is not noticeably slower than running the same program in the debugger with no breakpoints set.

The user who wishes to take advantage of this capability using a compiler other than VAX Ada should be aware that the compiler must provide adequate support for run-time symbolic debugging. This includes support for the equivalent of the VMS debugger "call," "deposit," and "set break" commands.

5.0 LINKED LISTS

EARS contains six linked list packages: Lnk, Lnk_AI, Lnk_TI, Link, Link_Ai, and Link_Ti. The linked list packages whose name begins with "Lnk" allow a user to store a copy of an object in a linked list or to retrieve a copy of an object from a linked list. These packages are instantiated with one generic parameter, that is of type ITEM. These packages do not support generic parameters that are of a limited private type.

The packages whose names begin with "Link" use an access type. This allows a user to manipulate objects in a linked list by pointing at them. These packages are instantiated with two parameters. One is of type REC, which is the designated type of the access type ITEM, and the other is of the access type ITEM. Rec can be a limited private type.

The linked list package names ending in "Ti" are used when the user wants to keep statistics on queue length that are not related to time. "Ti" refers to the SIMCRIPT tallied integer type.* Package names ending in "Ai" are used for maintaining time-dependent statistics on queue length, such as average buffer length. "Ai" refers to the SIMSCRIPT accumulated integer type.*

The specifications for Lnk and Link will be given with examples showing how to use them. The specifications for the remaining linked list packages are very similar to Lnk and Link and so is their use. Each linked list package will be described.

5.1 Lnk

Lnk provides subprograms that allow a user to store and retrieve copies of objects in a linked list.

5.1.1 Lnk Specification

The Lnk package specification is given below with detailed comments that explain how to use the subprograms. The user should review this specification.

```
--| ---------------------------------------------------
--| Instantiate "LNK" with an object of type "ITEM".
--| LNK defines an ordered linked list of type "LIST".
--| The ordering of the list is determined by the type of
--| operation used to insert Members into the list:
--|   ORDERING                  OPERATION
--|     FIFO                     Append
--|     LIFO                     Push
--|     BIFO                     Instantiation of "Incl"
--| ---------------------------------------------------

with Sim;

generic
  type ITEM is
    private;

package Lnk is
```

continued

*CACI, Inc. 1983. *SIMSCRIPT II.5 Programming Language*. Los Angeles, CA.

```
type LIST is
  private;

function Nl
  ( Lst : LIST )
    return NATURAL;

--| -----------------------------------------------
--| Returns the number of items in the list.
--| -----------------------------------------------

function Init
    return LIST;
--| -----------------------------------------------
--| Returns an initialized list.
--| This create a new linked list that contains 0 items.
--| -----------------------------------------------

procedure Dump
  ( Lst : in out LIST );
--| -----------------------------------------------
--| Dump removes all Items from a list.
--| -----------------------------------------------

procedure Dispose
  ( Lst : in out LIST );
--| -----------------------------------------------
--| Dump and Deallocate resources.
--| -----------------------------------------------

procedure Traverse
  ( Lst : in out LIST;
    Mbr : out ITEM;
    Found : out BOOLEAN;
    Dr : Sim.Dir := Sim.Normal );
--| -----------------------------------------------
--| Initializes a list traversal according to the DR
--| parameter. For nonempty lists, if "DR=Normal" then
--| the front element of "Lst" is returned in "Mbr" and
--| successive calls to "Next" and "Delete" return
--| elements successively closer to the back of "Lst".
--| If "DR=opposite", then the back element of "Lst" is
--| returned in "Mbr" and successive calls to "Next" and
--| "Delete" return elements successively closer to the
--| front of "Lst".
--|
--| If "Lst" is empty then the value returned in Mbr is
--| "null" and the value of "Found" is false.
--| -----------------------------------------------

procedure Next
  ( Lst : in out LIST;
    Mbr : out ITEM;
    Found : out BOOLEAN );

--------------------------------------------------
--| "Next" may only be called after "Traverse", "Next"
--| or "Delete". A call to "Next" after a call to any
--| other list operation is considered erroneous and may
--| cause unanticipated results. For nonempty lists,
--| "Next" returns a member, but does not remove it from
--| the "Lst".
--------------------------------------------------
procedure Delete
  ( Lst : in out LIST;
    Mbr : out ITEM;
    Found : out BOOLEAN );
```

continued

```
-----------------------------------------------------
--| "Delete" may only be called after "Traverse", "Next"
--| or "Delete". A call to "Delete" after a call to any
--| other list operation is considered erroneous and may
--| cause unanticipated results. For nonempty lists,
--| "Delete" returns a member "Mbr", and deletes from
--| the list the member returned in the previous call to
--| "Traverse", "Next" or "Delete". For example,
--| consider the following sequence:
--|     Traverse (L,M,F, Normal);
--|     Next (L,M,F);
--|     Delete (L,M,F);
--| The Call to "Delete" would have deleted the member
--| returned from the call to "Next".
-----------------------------------------------------

procedure Insert
  ( Lst : in out LIST;
    Mbr : ITEM );
-----------------------------------------------------
--| "Insert" operates relative to "Traverse", "Next",
--| and "Delete". If the direction is Normal, then
--| "Insert" inserts "Mbr" just before (i.e., toward
--| the front) the member returned from the last call
--| to "Traverse", "Next", or "Delete". If the
--| direction is Opposite then"Insert" inserts "Mbr"
--| just after (i.e., toward the back) the member
--| returned from the last call to "Traverse", "Next",
--| or "Delete".
-------------------------------------------------------

procedure Push
  ( Lst : in out LIST;
    Mbr : ITEM );
-----------------------------------------------------
--|          LIFO Queueing Discipline
-----------------------------------------------------
--| Procedure "Push" inserts a member at the front of
--| a list. It is equivalent to "Traverse, Insert"
--| with a direction of "Normal".
-----------------------------------------------------

procedure Append
  ( Lst : in out LIST;
    Mbr : ITEM );
-----------------------------------------------------
--|           FIFO Queueing Discipline
-----------------------------------------------------
--| Procedure "Append" Inserts a member at the Back of
--| a List. It is equivalent to "Traverse, Insert"
--| with a direction of "Opposite".
-----------------------------------------------------

procedure Pop
  ( Lst : in out LIST;
    Mbr : out ITEM;
    Found : out BOOLEAN );
-----------------------------------------------------
--| For nonempty lists, Pop deletes and returns the
--| front member of a list. If empty, it returns false
--| in "found".
-----------------------------------------------------

procedure Remove
  ( Lst : in out LIST;
    Mbr : ITEM );
-----------------------------------------------------
--| Procedure Remove searches for Mbr in the list. If
```

continued

```
--| it is found, it deletes it from the list.
----------------------------------------------------

generic
  type UNKNOWN is
    private;

----------------------------------------------------
--| Package search uses a user provided function that
--| defines the match. For example, the match may
--| be on one element in a record.
----------------------------------------------------
  with function Match
    ( Unk : UNKNOWN;
      Mbr : ITEM )
      return BOOLEAN;
package Search is

----------------------------------------------------
--| Procedure Find begins at the start of Lst to
--| search for Mbr that matches Unk. On a hit, Mbr is
--| returned and Found is set to true. User provided
--| match function doe the matching. Procedure Find
--| always returns first Mbr found.
----------------------------------------------------

  procedure Find
    ( Lst : in out LIST;
      Unk : UNKNOWN;
      Mbr : out ITEM;
      Found : out BOOLEAN );

----------------------------------------------------
--| Procedure Find_Next works like Find but is used
--| for subsequent calls to return next occurrences of
--| UNK.
----------------------------------------------------
  procedure Find_Next
    ( Lst : in out LIST;
      Unk : UNKNOWN;
      Mbr : out ITEM;
      Found : out BOOLEAN );

----------------------------------------------------
--| Procedure Remove starts at front of Lst and
--| removes 1st occurrence of Unk for which there is a
--| match, as defined in user provided match function.
----------------------------------------------------
  procedure Remove
    ( Lst : in out LIST;
      Unk : UNKNOWN;
      Mbr : out ITEM;
      Removed : out BOOLEAN );
end Search;

generic
  with function "<="
    ( Itm, Mbr : ITEM )
      return BOOLEAN is <>;
procedure Incl
  ( Lst : in out LIST;
    Itm : ITEM );
--| ------------------------------------------------
--| Searches from back to front. Mbr when Itm <= Mbr, Itm
```

continued

```
  --| is Inserted on the Rvs side of Mbr.
  --| -----------------------------------------------------

  Passed_End_Of_List, Uninitialized_List
    : exception;

-- Private portion of specification deleted in this
-- documentation.

end Lnk;
```

5.1.2 Lnk Example

The use of all subprograms that are defined in package Lnk is illustrated in the example below.

```
with Sim;
with Lnk;
with Text_IO;
procedure Test_Lnk is

  type TEST_NUMBER_TYPE is
    ( ONE, TWO );

  package Enum_IO is
    new Text_IO.Enumeration_IO (TEST_NUMBER_TYPE);

  type VAL_ARRAY is
    array (1 .. 10)
      of INTEGER;

  type T_REC is
    record
      Received : BOOLEAN;
      Tries : NATURAL;
      Del : FLOAT;
    end record;

  package T_Pack is
    new Lnk (T_REC);

  package I_Pack is
    new Lnk (INTEGER);

  T_List
    : T_Pack.List;

  Mbr
    : T_REC;

    -- Initialize members that are of type
    -- T_Rec.

  Mbr1
    : T_REC := (False, 1, 1.0);

  Mbr2
    : T_REC := (TRUE, 2, 2.0);
```

continued

```
Mbr3
  : T_REC := (False, 3, 3.0);

Mbr4
  : T_REC := (TRUE, 4, 4.0);

Mbr5
  : T_REC := (TRUE, 5, 5.0);

Mbr6
  : T_REC := (TRUE, 6, 6.0);

Mbr7
  : T_REC := (TRUE, 7, 7.0);

Mbr8
  : T_REC := (TRUE, 8, 8.0);

I_List
  : I_Pack.List;

L
  : INTEGER;

  -- Initialize members that are of type
  -- integer.

L1
  : INTEGER := 1;

L2
  : INTEGER := 2;

L3
  : INTEGER := 3;

L4
  : INTEGER := 4;

Fnd
  : BOOLEAN;

Removed
  : BOOLEAN;

Test_Numb
  : TEST_NUMBER_TYPE;

No_Of_Items
  : NATURAL;

Unk
  : INTEGER;

  -- Function below used for generic function MATCH in
  -- package SEARCH.
  -- When N and M match function A returns true.

function A
  ( N : INTEGER;
    M : INTEGER )
    return BOOLEAN is
begin -- A
  if N = M then
```

continued

```
      return TRUE;
    else
      return False;
    end if;
  end A;

  package S_Pack is
    new I_Pack.Search (INTEGER, A);

    -- Function below is imported for generic
    -- function "<=" which is used with procedure INC.

  function Less_Important
    ( L : INTEGER;
      R : INTEGER )
      return BOOLEAN is
  begin -- Less_Important
    if L > R then
      return TRUE;
    else
      return False;
    end if;
  end Less_Important;

  procedure Include is
    new I_Pack.Incl (Less_Important);


begin -- Test_Lnk
  Text_IO.Put("Enter test number >");
  Enum_IO.Get(Test_Numb);
  case Test_Numb is
    when ONE =>
    -- Init initializes the linked list whose members
    -- are T_Recs.
      T_List := T_Pack.Init;
      -- Next line positions to front
      -- of T_List. It is an empty
      -- linked list so Mbr is null and
      -- Fnd is false.
      T_Pack.Traverse(T_List, Mbr, Fnd);
      -- Next line inserts Mbr1 in
      -- in T_List.
      T_Pack.Insert(T_List, Mbr1);
      -- Next two lines insert Mbr2
      -- in front of Mbr1 and then
      -- Mbr3 in front of Mbr2.
      T_Pack.Insert(T_List, Mbr2);
      T_Pack.Insert(T_List, Mbr3);
      -- Next two lines insert Mbr4
      -- at end of list and then
      -- Mbr5 after Mbr4.
      T_Pack.Append(T_List, Mbr4);
      T_Pack.Append(T_List, Mbr5);
      -- Next line removes Mbr1 from list.
      T_Pack.Remove(T_List, Mbr1);
      -- Number of items in list is 4.
      No_Of_Items := T_Pack.Nl(T_List);
      -- The order of T_List now is:
      --    Mbr3
      --    Mbr2
      --    Mbr4
      --    Mbr5
```

continued

```
        -- Dump clears all members from list.
        T_Pack.Dump(T_List);
        -- Number of items in list is 0.
        No_Of_Items := T_Pack.Nl(T_List);
        -- Next four statements insert members so in
        -- sequential order in list.
        T_Pack.Push(T_List, Mbr2);
        T_Pack.Push(T_List, Mbr1);
        T_Pack.Append(T_List, Mbr3);
        T_Pack.Append(T_List, Mbr4);
        -- Next statement positions to end of list.
        T_Pack.Traverse(T_List, Mbr, Fnd, Sim.Opposite);
        -- Move to second member from end, Mbr3.
        T_Pack.Next(T_List, Mbr, Fnd);
        -- Delete Mbr3 from list.
        T_Pack.Delete(T_List, Mbr, Fnd);
        -- Remove front member from list.
        -- Front member is Mbr1.
        T_Pack.Pop(T_List, Mbr, Fnd);
        -- No_of_Items below is two.
        No_Of_Items := T_Pack.Nl(T_List);
        -- Dispose clears list and deallocates
        -- resources.
        T_Pack.Dispose(T_List);
      when TWO =>
      -- Init initializes the linked list whose members
      -- are of type integer.
        I_List := I_Pack.Init;
        -- No_of_Items in list is 0.
        No_Of_Items := I_Pack.Nl(I_List);
        I_Pack.Push(I_List, L4);
        I_Pack.Push(I_List, L3);
        I_Pack.Push(I_List, L1);
        -- Next statement starts at end of I_List and
        -- inserts L2 in front of first member of
        -- list that it is less than.
        Include(I_List, L2);
        -- After INCLUDE the list is ordered:
        --     L1
        --     L2
        --     L3
        --     L4
        --
        -- Find searches I_List from front for first
        -- Mbr that matches UNK. Match is defined in
        -- user provided function A above.
        Unk := 3;
        S_Pack.Find(I_List, Unk, L, Fnd);
        -- The value returned above for L is 3 and
        -- Fnd is true.

        -- Next statement searches for next match.
        -- In this case there are none and Fnd is false.
        S_Pack.Find_Next(I_List, Unk, L, Fnd);
        -- Next statement removes member
        -- that matches UNK. Again match is defined in
        -- user provided function A.
        -- Values returned in L and removed are 3 and
        -- true respectively.
        S_Pack.Remove(I_List, Unk, L, Removed);
    end case;
  end Test_Lnk;
```

5.2 Lnk_Ai

Package Lnk_Ai closely resembles package Lnk. In addition to containing the same subprograms as Lnk, package Lnk_A offers several additional capabilities. For example, Lnk_A maintains accumulated integer statistics on queue length and provides a method for identifying variables that may be graphed. The graphing capability has not yet been implemented. The accumulated integer statistics on queue length are maintained automatically. To examine the variable of statistics associated with a list, LIST, the user calls the function Var. The specification for Var is

```
function Var
  (Lst : LIST )
  return Ai.Variable;
```

The statistics variable, Ai.Variable, is explained in Section 6.7.

The discriminant in the type statement, shown below, is a sequence number that identifies the variable that might be graphed. The sequence number is incremented when the variable is declared.

```
type LIST
  (Seq :NATURAL := Stat.Seq(Stat.Ai) is
  private;
```

5.3 Lnk_Ti

Package Lnk_Ti provides a capability for (1) storing and retrieving a copy of an object in a linked list and (2) maintaining integer statistics on queue length.

5.4 Link

Package Link provides a capability for storing and retrieving objects in a linked list when using an access type.

5.4.1 Link Specification

The package specification for Link is given, with comments, below.

```
----------------------------------------------------------
--| Instantiate "LINK" with an object "Rec" and its access
--| type "ITEM". LINK defines "LIST", an ordered linked
--| list.  The ordering of the list is determined by the
--| type of operation used to insert Members into the

                                                continued
```

```
--| list:
--|   ORDERING              OPERATION
--|     FIFO                  Append
--|     LIFO                  Push
--|     BIFO                  instantiation of "Incl"
----------------------------------------------------------

with Sim;
generic
  type REC is
    limited private;

  type ITEM is
    access REC;

package Link is

  type LIST is
    private;

  function Nl
    ( Lst : LIST )
      return NATURAL;
  --| ----------------------------------------------------
  --| Returns the number of items in the list.
  --| ----------------------------------------------------

  function Init
      return LIST;
  --| ----------------------------------------------------
  --| Returns an initialized list.
  --| ----------------------------------------------------

  function First
    ( Lst : LIST )
    return ITEM;
  --| ----------------------------------------------------
  --| Returns the first member in the list (does not
  --| delete).
  --| ----------------------------------------------------

  procedure Dump
    ( Lst : in out LIST );
  --| ----------------------------------------------------
  --| Dump removes all Items from a list.
  --| ----------------------------------------------------

  procedure Traverse
    ( Lst : in out LIST;
      Mbr : out ITEM;
      Found : out BOOLEAN;
      Dr : Sim.Dir := Sim.Normal );
  --------------------------------------------------------
  --| Initializes a list traversal according to the DR
  --| parameter. For nonempty lists, if "DR=Normal" then
  --| the front element of "Lst" is returned in "Mbr" and
  --| successive calls to "Next" and "Delete" return
  --| elements successively closer to the back of "Lst". If
  --| "DR=opposite", then the back element of "Lst" is
  --| returned in "Mbr" and successive calls to "Next" and
  --| "Delete" returns elements successively closer to the
  --| front of "Lst".
  --| If "Lst" is empty then the value returned in Mbr is
```

continued

```
--| "null" and the value of "Found" is false.
-----------------------------------------------------

procedure Next
  ( Lst : in out LIST;
    Mbr : out ITEM;
    Found : out BOOLEAN );
-----------------------------------------------------
--| "Next" may be called after "Traversal", "Next" or
--| "Delete". A call to "Next" after a call to any other
--| list operation is considered erroneous and may cause
--| unanticipated results. For nonempty lists, "Next"
--| returns a member, but does not remove it from the
--| "Lst".
-----------------------------------------------------

procedure Delete
  ( Lst : in out LIST;
    Mbr : out ITEM;
    Found : out BOOLEAN );
-----------------------------------------------------
--| "Delete" may only be called after "Traverse", "Next"
--| or "Delete". A call to "Delete" after a call to any
--| other list operation is considered erroneous and may
--| cause unanticipated results. For nonempty lists,
--| "Delete" returns a member "Mbr", and deletes from
--| the list the member returned in the previous call to
--| "Traverse", "Next" or "Delete". For example, consider
--| the following sequence:
--|     Traverse (L,M,F, Normal);
--|     Next (L,M,F);
--|     Delete (L,M,F);
--| The Call to "Delete" would have deleted the member
--| returned from the call to "Next".
-----------------------------------------------------

procedure Insert
  ( Lst : in out LIST;
    Mbr : ITEM );
-----------------------------------------------------
--| "Insert" operates relative to "Traverse", "Next", and
--| "Delete". If the direction is Normal, then "Insert"
--| insert "Mbr" just before (i.e., toward the front) the
--| member returned from the last call to "Traverse",
--| "Next", or "Delete". If the direction is Opposite
--| then "Insert" inserts "Mbr" just after (i.e., toward
--| the back) the member returned from the last call to
--| "Traverse", "Next", or "Delete".
-----------------------------------------------------

procedure Push
  ( Lst : in out LIST;
    Mbr : ITEM );
-----------------------------------------------------
--|          LIFO Queueing Discipline
-----------------------------------------------------
--| Procedure "Push" inserts a member at the front of a
--| list. It is equivalent to "Traverse, Insert" with a
--| direction of "Normal".
-----------------------------------------------------

procedure Append
  ( Lst : in out LIST;
    Mbr : ITEM );
```

continued

```
-----------------------------------------------------------
--|           FIFO Queueing Discipline
-----------------------------------------------------------
--| Procedure "Append" Inserts a member at the Back of a
--| List. It is equivalent to "Traverse, Insert" with a
--| direction of "Opposite".
-----------------------------------------------------------

procedure Pop
  ( Lst : in out LIST;
    Mbr : out ITEM;
    Found : out BOOLEAN );
-----------------------------------------------------------
--| For nonempty lists, Pop deletes and returns the front
--| member of a list. If empty, it returns false in
--| "found".
-----------------------------------------------------------

procedure Remove
  ( Lst : in out LIST;
    Mbr : ITEM );
-----------------------------------------------------------
--| Procedure Remove searches for Mbr in the list. If it
--| is found, it deletes it from the list.
-----------------------------------------------------------

generic
  type UNKNOWN is
    private;

-----------------------------------------------------------
--| Package search uses a user provided function that
--| defines the match. For example, the match may
--| be on one element in a record.
-----------------------------------------------------------
  with function Match
    ( Unk : UNKNOWN;
      Mbr : ITEM )
      return BOOLEAN;
package Search is

-----------------------------------------------------------
--| Procedure Find begins at the start of Lst to search
--| for Mbr that matches Unk. On a hit, Mbr is returned
--| and Found is set to true. User provided match
--| function is provided for matching. Procedure Find
--| always returns first Mbr found.
-----------------------------------------------------------
  procedure Find
    ( Lst : in out LIST;
      Unk : UNKNOWN;
      Mbr : out ITEM;
      Found : out BOOLEAN );

-----------------------------------------------------------
--| Procedure Find_Next works like Find but is used for
--| subsequent calls to return next occurrences of UNK.
-----------------------------------------------------------
  procedure Find_Next
    ( Lst : in out LIST;
      Unk : UNKNOWN;
      Mbr : out ITEM;
      Found : out BOOLEAN );
```

continued

```
------------------------------------------------------
--| Procedure Remove starts at front of Lst and removes
--| 1st occurrence of Unk for which there is a match,
--| as defined in user provide match function.
------------------------------------------------------
  procedure Remove
    ( Lst : in out LIST;
      Unk : UNKNOWN;
      Mbr : out ITEM;
      Removed : out BOOLEAN );
end Search;

generic
  with function "<="
    ( Itm, Mbr : ITEM )
      return BOOLEAN is <>;
procedure Incl
  ( Lst : in out LIST;
    Itm: ITEM );
--| ----------------------------------------------------
--| Searches from back to front. Mbr when Itm <= Mbr, Itm
--| is Inserted on the Rvs side of Mbr.
--| ----------------------------------------------------

generic
  with procedure Free
    ( Mbr : in out ITEM );
procedure Destroy
  ( Lst : in out LIST );
--| ----------------------------------------------------
--| This routine Dumps and FREEs every element in the list,
--| then "uninitializes" the list.
--| ----------------------------------------------------

Null_Item, Passed_End_Of_List, Uninitialized_List
  : exception;

-- Private portion of specification deleted in this
-- document.

end Link;
```

5.4.2 Link Example

The example below illustrates the use of the subprograms contained in the package specification Link. This example closely resembles the example given in Section 5.1.2.

```
with Unchecked_Deallocation;
with Sim;
with Link;
with Text_IO;
procedure Test_Link is

  type TEST_NUMBER_TYPE is
```

continued

```
  ( ONE, TWO );

package Enum_IO is
  new Text_IO.Enumeration_IO (TEST_NUMBER_TYPE);

type VAL_ARRAY is
  array (1 .. 10)
    of INTEGER;

type T_REC is
  record
    Received : BOOLEAN;
    Tries : NATURAL;
    Del : FLOAT;
  end record;

type T_PTR is
  access T_REC;

package T_Pack is
  new Link (T_REC, T_PTR);

type IP is
  access INTEGER;

package I_Pack is
  new Link (INTEGER, IP);

T_List
  : T_Pack.List;

  -- Define members that are of access type.

Mbr
  : T_PTR;

Mbr1
  : T_PTR;

Mbr2
  : T_PTR;

Mbr3
  : T_PTR;

Mbr4
  : T_PTR;

Mbr5
  : T_PTR;

Mbr6
  : T_PTR;

Mbr7
  : T_PTR;

Mbr8
  : T_PTR;

Unk2
  : T_PTR;

I_List
```

continued

```
  : I_Pack.List;

  -- Define members that are of type
  -- integer.

L1
  : IP;

L2
  : IP;

L3
  : IP;

L4
  : IP;

L
  : IP;

Fnd
  : BOOLEAN;

Removed
  : BOOLEAN;

Test_Numb
  : TEST_NUMBER_TYPE;

No_Of_Items
  : NATURAL;

Unk
  : INTEGER;

  -- Procedure Free is input to generic procedure Destroy.
  -- Destroy dumps and frees every element in a list and
  -- then "unitializes" the list.

procedure Deallocate is
  new Unchecked_Deallocation (T_REC, T_PTR);

procedure Free
  ( P : in out T_PTR )
  renames Deallocate;

procedure Destroyer is
  new T_Pack.Destroy (Free);

  -- Function A is used for generic function MATCH in
  -- package SEARCH. When N and M match, function A
  -- returns true.

function A
  ( N : INTEGER;
    M : IP )
    return BOOLEAN is
begin -- A
  return N = M.all;
end A;

-- Function B is used for generic function MATCH in
-- package SEARCH. When N and M match, function A
```

continued

```
    -- returns true.
    function B
      ( N : T_PTR;
        M : T_PTR )
        return BOOLEAN is
    begin -- B
      if N.Del = M.Del then
        return TRUE;
      else
        return False;
      end if;
    end B;

    package S_Pack is
      new I_Pack.Search (INTEGER, A);

    package U_Pack is
      new T_Pack.Search (T_PTR, B);


      -- Function below is imported for generic
      -- function "<=" which is used with procedure INC.

    function Less_Important
      ( L : IP;
        R : IP )
        return BOOLEAN is
    begin -- Less_Important
      return L.all > R.all;
    end Less_Important;

    procedure Include is
      new I_Pack.Incl (Less_Important);


  begin -- Test_Link

  -- Initialize members.
    Mbr := new T_REC;
    Mbr1 := new T_REC'(False, 1, 1.0);
    Mbr2 := new T_REC'(TRUE, 2, 2.0);
    Mbr3 := new T_REC'(False, 3, 3.0);
    Mbr4 := new T_REC'(TRUE, 4, 4.0);
    Mbr5 := new T_REC'(TRUE, 5, 5.0);
    Mbr6 := new T_REC'(TRUE, 6, 6.0);
    Mbr7 := new T_REC'(TRUE, 7, 7.0);
    Mbr8 := new T_REC'(TRUE, 1, 8.0);
    Unk2 := new T_REC'(TRUE, 1, 1.0);

    L1 := new INTEGER'(1);
    L2 := new INTEGER'(2);
    L3 := new INTEGER'(3);
    L4 := new INTEGER'(4);
    L := new INTEGER;

    Text_IO.Put("Enter test number >");
    Enum_IO.Get(Test_Numb);
    case Test_Numb is
      when ONE =>
      -- Init initializes the linked list whose members
      -- are T_Recs.
        T_List := T_Pack.Init;
        -- Next line positions to front
        -- of T_List. It is an empty
```

continued

```
    -- linked list so Mbr is null and
    -- Fnd is false.
    T_Pack.Traverse(T_List, Mbr, Fnd);
    -- Next line inserts Mbr1 in
    -- in T_List.
    T_Pack.Insert(T_List, Mbr1);
    -- Next two lines insert Mbr2
    -- in front of Mbr1 and then
    -- Mbr3 in front of Mbr2.
    T_Pack.Insert(T_List, Mbr2);
    T_Pack.Insert(T_List, Mbr3);
    -- Next two lines insert Mbr4
    -- at end of list and then
    -- Mbr5 after Mbr4.
    T_Pack.Append(T_List, Mbr4);
    T_Pack.Append(T_List, Mbr5);
    -- Next line removes Mbr1 from list.
    T_Pack.Remove(T_List, Mbr1);
    -- Number of items in list is 4.
    No_Of_Items := T_Pack.Nl(T_List);
    -- The order of T_List now is:
    --   Mbr3
    --   Mbr2
    --   Mbr4
    --   Mbr5
    -- Dump clears all members from list.
    T_Pack.Dump(T_List);
    -- Number of items in list is 0.
    No_Of_Items := T_Pack.Nl(T_List);
    -- Next four statements insert members so in
    -- sequential order in list.
    T_Pack.Push(T_List, Mbr2);
    T_Pack.Push(T_List, Mbr1);
    T_Pack.Append(T_List, Mbr3);
    T_Pack.Append(T_List, Mbr4);
    -- Next statement positions to end of list.
    T_Pack.Traverse(T_List, Mbr, Fnd, Sim.Opposite);
    -- Move to second member from end, Mbr3.
    T_Pack.Next(T_List, Mbr, Fnd);
    -- Delete Mbr3 from list.
    T_Pack.Delete(T_List, Mbr, Fnd);
    -- Remove front member from list.
    -- Front member is Mbr1.
    T_Pack.Pop(T_List, Mbr, Fnd);
    -- No_of_Items below is two.
    No_Of_Items := T_Pack.Nl(T_List);
    -- Get first member in list.
    Mbr := T_Pack.First(T_List);
    -- See if next member matches UNK2;
    U_Pack.Find(T_List, Unk2, Mbr, Fnd);
    -- Dump and free elements in T_List.
    -- Then "uninitialize" T_List.
    Destroyer(T_List);
  when TWO =>
  -- Init initializes the linked list whose members
  -- are of type integer.
    I_List := I_Pack.Init;
    -- No_of_Items in list is 0.
    No_Of_Items := I_Pack.Nl(I_List);
    I_Pack.Push(I_List, L4);
    I_Pack.Push(I_List, L3);
    I_Pack.Push(I_List, L1);
    -- Next statement starts at end of I_List and
    -- inserts L2 in front of first member of
```

continued

```
      -- list that it is less than.
      Include(I_List, L2);
      -- After INCLUDE the list is ordered:
      --     L1
      --     L2
      --     L3
      --     L4
      --
      -- Find searches I_List from front for first
      -- Mbr that matches UNK. Match is defined in
      -- user provided function A above.
      Unk := 3;
      S_Pack.Find(I_List, Unk, L, Fnd);
      -- The value returned above for L is 3 and
      -- Fnd is true.

      -- Next statement searches for next match.
      -- In this case there are none and Fnd is false.
      S_Pack.Find_Next(I_List, Unk, L, Fnd);
      -- Next statement removes member
      -- that matches UNK. Again match is defined in
      -- user provided function A.
      -- Values returned in L and removed are 3 and
      -- true respectively.
      S_Pack.Remove(I_List, Unk, L, Removed);
  end case;
end Test_Link;
```

5.5 Link_Ai

Package Link_Ai is similar to package Link. Link_Ai provides the additional capability of maintaining accumulated integer statistics on queue length and a means for identifying variables that might, in the future, be graphed.

The function below (as in package Lnk_Ai, Section 5.2) allows the user to get the variable that contains the statistics on queue length.

```
function Var
  (Lst : LIST )
  return Ai.Variable;
```

The discriminant for the type statement for the list (also as in package Lnk_Ai) is a sequence number that identifies a variable that might be graphed.

5.6 Link_Ti

Package Link_Ti is also similar to package Link. Link_Ti provides the additional capability of maintaining tallied integer statistics on queue length. The sequence number for graphing is also kept as a discriminant in the "type LIST" statement.

6.0 STATISTICAL PACKAGES

EARS contains seven primary statistical packages. The first three that will be discussed, package Rand, Random_Distribution, and Random, are used for random number generation. The remaining, packages, Ti, Tf, Ai, and Af, are used for maintaining and reporting tallied integer, tallied floating point, accumulated integer, and accumulated floating point statistics. The terms "tallied" and "accumulated" are defined in the C.A.C.I. publication.* The equations used to calculate the tallied and accumulated mean and variance are given in the package specifications for Ti, Tf, Ai, and Af.

EARS also contains package Stat. When writing a simulation the user does not need to write any code that uses anything in this package. Therefore this package specification will not be included in this guide.

6.1 PACKAGE Rand SPECIFICATION

The package Rand specification, shown below, contains three subprograms. The first, function Number, returns a pseudorandom number between 0.0 and 1.0. The next two, procedure Save_Streams and Restore_Streams, save and restore the values in the random number streams.

```
with Sim;

package Rand is

  subtype STREAM_RANGE is
    INTEGER range 1 .. 100;

  -- Number returns a random number using a seed found in the    -- designated
stream.
  function Number
    ( Stream : STREAM_RANGE )
      return Sim.Real;

  -- Save_Streams writes the stream of seeds to the disk
  -- file File_Name.
  procedure Save_Streams
    ( File_Name : STRING );

  -- Restore_Streams reads in the streams from disk file
  -- File_Name.
  procedure Restore_Streams
    ( File_Name : STRING );

end Rand;
```

6.2 PACKAGE Random_Distribution SPECIFICATION

Package Random_Distribution, given below, contains functions that allow a user to generate random numbers from uniform, exponential, normal, and poisson distributions, as well as from a step function. In EARS, package Random_Distribution is instantiated in the package Random specification, which is given in the next section.

* CACI, Inc. 1983. *SIMSCRIPT II.5 Programming Language.* Los Angeles, CA.

```
 with Sim;
 use Sim;

 generic
  -- STREAM_RANGE ranges from 1 to the maximum
  -- number of streams.
  type STREAM_RANGE is
    range <>;

  -- Function Produce_Random returns a random
  -- generator seed.
  with function Produce_Random
    ( Stream : STREAM_RANGE )
      return Real;
package Random_Distribution is

  function Uniform
    ( Stream : STREAM_RANGE;
      A : Real := 0.0;
      B : Real := 1.0 )
      return Real;

  function Exponential
    ( Stream : STREAM_RANGE;
      Lambda : Real )
      return Real;

  function Normal
    ( Stream : STREAM_RANGE;
      Mean : Real;
      Standard_Deviation : Real )
      return Real;

  function Poisson
    ( Stream : STREAM_RANGE;
      Lambda : Real )
      return Real;

  -- Function Step returns a uniformly distributed
  -- random number, that is an integer, in the
  -- range Low to Hi.
  function Step
    ( Stream : STREAM_RANGE;
      Low, Hi : INTEGER )
      return INTEGER;

end Random_Distribution;
```

6.3 PACKAGE Random SPECIFICATION

This specification, shown below, contains the instantiation of Random_Distribution that is used for EARS simulations. The stream range is from 1 to 100 for the VAX/VMS random number generator. A user may supply his or her own random number generator that has a different stream range. The function Number in package Rand is used to provide the seed.

```
with Rand, Random_Distribution;
package Random is
  new Random_Distribution (Stream_Range => Rand.Stream_Range,
    Produce_Random => Rand.Number);
```

6.4 PACKAGE Rand and Random_Distribution EXAMPLE

The example below illustrates how these random distribution packages are used.

```
with Random;
with Sim;
with Rand;
procedure Example_Random is

   Time1 : Sim.Real;
   Time2 : Sim.Real;
   Time3 : Sim.Real;
   Time4 : Sim.Real;
   Time5 : integer;

begin
   -- The next 5 executable statements illustrate how random
   -- numbers from different distributions are obtained when
   -- Rand.Number is the procedure that provides the random
   -- number. Remember, this is true because in package
   -- Random the generic package  Random_Distribution is
   -- instantiated such that function Rand.Number returns
   -- the seed.

   -- Get a random variable from a uniform distribution that
   -- ranges from 0.0 to 5.0.
   -- A seed from stream 1 is provided by function
   -- Rand.Number
   Time1 := Random.Uniform(1,0.0,5.0);
                           -- Time1 = 1.73500448465347

   -- Get a random variable from a uniform distribution that
   -- ranges from 0.0 to 1.0.
   -- A seed from stream 1 is provided by function
   -- Rand.Number
   Time2 := Random.Uniform(1);
                           -- Time2 = 7.5552463531494142-003

   -- Get a random variable from an exponential
   -- distribution whose mean is 0.5.
   -- A seed from stream 1 is provided by function
   -- Rand.Number
   Time3 := Random.Exponential(1,0.5);
                           -- Time3 = 8.9373453088236332-002

   -- Get a random variable from a normal distribution
   -- whose mean is 10.0 and standard deviation is 1.0.
   -- Same seed as above.
   Time4 := Random.Normal(1,10.0,1.0);
                           -- Time4 = 10.5573352087283

   -- Get a random variable from a uniform distribution that
   -- ranges from 0 to 99. The random variable is an integer.
   -- Same seed as above.
   Time5 := Random.Step(1,0,99);
                           -- Time5 = 0

   -- The next statement illustrate the use of procedure
   -- Rand.Save_Streams which saves the current stream of
   -- random number seeds that is defined in package body
   -- Rand. They are saved in file Stream_File1.
   Rand.Save_Streams("Stream_File1");

   -- The next statement reads this stream from file
   -- Stream_File1, back into package body Rand.
   Rand.Restore_Streams("Stream_File1");
end Example_Random;
```

6.5 PACKAGE Ti SPECIFICATION

The specification for package Ti is given below with comments. This specification defines a variable type for maintaining statistical information and provides procedures required for maintaining and reporting tallied integer statistics. The procedure Reset is used to reinitialize the values in the user's parameter that is of type VARIABLE. This is only done to reinitialize the values. The initialization of these values is automatic to the user. Procedure Assign is used to continuously update the statistical information. Procedures Mean, Variance, and Std_Dev calculate and return these statistics. Procedure Report writes out these statistics and additional information to either the user's monitor or to a file. Procedure Brief writes out statistics and additional information that is needed for graphing to a file. Procedure Report writes descriptive words (in strings) next to the data, so that the file is easy to read. Procedure Brief only writes out the data.

When using procedures Mean, Variance, Std_Dev, and Report, the user has the option of either having exceptions raised within the procedure or having an Error_Type returned to the calling procedure.

```
with Sim, Text_IO, Stat;
use Sim, Text_IO;

package Ti is

  -- Error_Type contains types of errors that may be
  -- propagated from the subprograms in this
  -- package.
  type ERROR_TYPE is
    ( NONE, CONSTRAINT, NODATA, OTHER );

  -- Variable is the type definition for records
  -- where statistical information is stored.
  -- The discriminant, Seq, contains an ID number
  -- that is intended to be used for graphing
  -- purposes. It may, for example, be used as a
  -- node ID. Procedure Brief, below, writes
  -- statistics to a file which later may be graphed.
  type VARIABLE
    ( Seq : NATURAL := Stat.Seq(Stat.Ti) ) is
    record
      N : NATURAL := 0;
      Sum_X : NATURAL := 0;
      Sn : Real := 0.0;
      Max_X : INTEGER := INTEGER'First;
      Min_X : INTEGER := INTEGER'Last;
      X : INTEGER;
    end record;

  procedure Reset
    ( Var : in out VARIABLE );
  --| Resets Stats;

  procedure Assign
    ( Var : in out VARIABLE;
      Value : INTEGER );
  --| Assign a new value to the variable.
```

continued

```
  -- The equation below is used to calculate the mean.
  --
  --  mu(n) = sum of n values/n
  --
  --    where n represents the number of observations and
  --          what is inside parentheses is a subscript
  --
  -- The equation below is used to calculate the variance.
  --
  -- S(n+1) = S(n) + [n/[n+1]] * [x(n+1) - mu(n)]**2   --
  -- V(n+1) = [1/n] * S(n+1)

  procedure Mean
    ( Var : VARIABLE;
      Result : out Real;
      Error : out ERROR_TYPE;
      Propagate : BOOLEAN := TRUE );
  --| Return the average value of the variable.
  --| Exceptions may or may not be propagated.

  procedure Variance
    ( Var : VARIABLE;
      Result : out Real;
      Error : out ERROR_TYPE;
      Propagate : BOOLEAN := TRUE );
  --| Return the variance of the variable values.
  --| Exceptions may or may not be propagated.

  procedure Std_Dev
    ( Var : VARIABLE;
      Result : out Real;
      Error : out ERROR_TYPE;
      Propagate : BOOLEAN := TRUE );
  --| Return the standard deviation of the variable values.
  --| Exceptions may or may not be propagated.

  procedure Report
    ( Var : VARIABLE;
      Propagate : BOOLEAN := False );
  --| Prints statistics, in human readable form, to monitor.
  --| Exceptions may or may not be propagated.

  procedure Report
    ( F : File_Type;
      Var : VARIABLE;
      Propagate : BOOLEAN := False );
  --| Prints statistics, in human readable form, to a file.
  --| Exceptions may or may not be propagated.

  procedure Brief
    ( F : File_Type;
      Typ : Stat.Id;
      Var : VARIABLE );
  --| Writes Brief report to a file according to format
  --| specifications defined in package Stat. File contains
  --| statistics and is intended for graphing purposes.
  --| Typ is a 4 character string ID provided by the user
end Ti;
```

6.6 PACKAGE Tf SPECIFICATION

This specification, shown below, defines a variable type for maintaining statistical information and provides procedures required for maintaining and reporting tallied floating point statistics. The package Tf specification, which is given below, is similar to the package Ti specification. The comments explain the relevant aspects of this specification.

```
with Sim, Text_IO, Stat;
use Sim, Text_IO;

package Tf is

  -- Error_Type contains types of errors that may be
  -- propagated up from the subprograms in this
  -- package.
  type ERROR_TYPE is
    ( NONE, CONSTRAINT, NODATA, OTHER );

  -- Variable is the type definition for records
  -- where statistical information is stored.
  -- The discriminant Seq is a sequence number that is
  -- an ID number that is needed for graphing purposes.
  -- Procedure Brief writes statistics to a file
  -- which later may be graphed.
  type VARIABLE
    ( Seq : NATURAL := Stat.Seq(Stat.Tf) ) is
    record
      N : INTEGER := 0;
      Sum_X : Real := 0.0;
      Sn : Real := 0.0;
      Max_X : Real := Real'First;
      Min_X : Real := Real'Last;
      X : Real;
    end record;

  procedure Reset
    ( Var : in out VARIABLE );
  --| Resets Stats;

  procedure Assign
    ( Var : in out VARIABLE;
      Value : Real );
  --| Assign a new value to the variable.

  -- The equation below is used to calculate the mean.
  --
  --  mu(n) = sum of n values/n
  --
  --    where n represents the number of observations and
  --          what is inside parentheses is a subscript
  --
  -- The equation below is used to calculate the variance.
  --
  -- S(n+1) = S(n) + [n/[n+1]] * [x(n+1) - mu(n)]**2
  --
  -- V(n+1) = [1/n] * S(n+1)



  procedure Mean
    ( Var : VARIABLE;
      Result : out Real;
      Error : out ERROR_TYPE;
      Propagate : BOOLEAN := TRUE );
  --| Return the average value of the variable.
  --| Exceptions may or may not be propagated.

  procedure Variance
    ( Var : VARIABLE;
      Result : out Real;
      Error : out ERROR_TYPE;
      Propagate : BOOLEAN := TRUE );
  --| Return the variance of the variable values.
```

continued

```
  --| Exceptions may or may not be propagated.

  procedure Std_Dev
    ( Var : VARIABLE;
      Result : out Real;
      Error : out ERROR_TYPE;
      Propagate : BOOLEAN := TRUE );
  --| Return the standard deviation of the variable values.
  --| Exceptions may or may not be propagated.

  procedure Report
    ( Var : VARIABLE;
      Propagate : BOOLEAN := False );
  --| Prints statistics, in human readable form, to monitor.
  --| Exceptions may or may not be propagated.

  procedure Report
    ( F : File_Type;
      Var : VARIABLE;
      Propagate : BOOLEAN := False );
  --| Prints statistics, in human readable form, to a file.
  --| Exceptions may or may not be propagated.

  procedure Brief
    ( F : File_Type;
      Typ : Stat.Id;
      Var : VARIABLE );
  --| Writes Brief report to a file according to format
  --| specifications defined in package Stat. File contains
  --| statistics and is intended for graphing purposes.
  --| Typ is a 4 character string ID provided by the user.
end Tf;
```

6.7 PACKAGE Ai SPECIFICATION

This specification defines a variable type used for maintaining statistical information and provides the procedures required for maintaining and reporting accumulated integer statistics. This specification is analogous to the package Ti specification. The specification with comments is given below.

```
with Text_IO, Stat;
use Text_IO;
with Sim;
use Sim;

----------------------------------------------------------
package Ai is
----------------------------------------------------------

  -- Error_Type contains types of errors that may be
  -- propagated from the subprograms in this package
  type ERROR_TYPE is
    ( NONE, CONSTRAINT, NODATA, OTHER );
```

continued

```
-- Variable is the type definition for records
-- where statistical information is stored.
-- The discriminant Seq is a sequence number
-- that is in an ID number used for graphing
-- purposes. Procedure Brief writes statistics
-- for a file which later may be graphed.
type VARIABLE
  ( Seq : NATURAL := Stat.Seq(Stat.Ai) ) is
  record
    Initialized : boolean := false;
    T_0 : Real := 0.0;
    T_Last : Real := 0.0;
    Sum_X : Real := 0.0;
    Sum_Squares_X : Real := 0.0;
    Max_X : INTEGER := INTEGER'First;
    Min_X : INTEGER := INTEGER'Last;
    X : INTEGER;
  end record;

procedure Reset
  ( Var : in out VARIABLE );
-- Reset Stats.

procedure Assign
  ( Var : in out VARIABLE;
    Value : INTEGER );
-- Assign a new value to this variable.

-- The equations below, which came from the text,
-- "SIMSCRIPT II.5 Programming Language", C.A.C.I.,
-- (pg 390), are used to calculate the mean
-- and variance.
--
-- SUM              summation of x * [TIME.V - TL]
-- SUM.OF.SQUARES   summation of [x**2] * [TIME.V -TL]
-- MEAN             SUM/[TIME.V - T0]
-- MEAN.SQUARE      SUM.OF.SQUARES/[TIME.V - T0]
-- VARIANCE         MEAN.SQUARE  -  MEAN**2
--
--     where TIME.V is simulation clock time,
--           TL is the simulated time at which an
--              an accumulated variable was set to
--              its current value, and
--           T0 is the simulated time at which
--              accumulation starts

procedure Mean
  ( Var : VARIABLE;
    Result : out Real;
    Error : out ERROR_TYPE;
    Propagate : BOOLEAN := TRUE );
-- Return the mean value of the variable.
-- Exceptions may or may not be propagated.

procedure Variance
  ( Var : VARIABLE;
    Result : out Real;
    Error : out ERROR_TYPE;
    Propagate : BOOLEAN := TRUE );
-- Return the variance of the variable values;
-- Exceptions may or may not be propagated.

procedure Std_Dev
  ( Var : VARIABLE;
```

continued

```
      Result : out Real;
      Error : out ERROR_TYPE;
      Propagate : BOOLEAN := TRUE );
  -- Return the standard deviation of the values.
  -- Exceptions may or may not be propagated.

  procedure Report
    ( Var : VARIABLE;
      Propagate : BOOLEAN := False );
  -- Prints statistics, in human readable form, to monitor.
  -- Exceptions may or may not be propagated.

  procedure Report
    ( F : File_Type;
      Var : VARIABLE;
      Propagate : BOOLEAN := False );
  -- Prints statistics, in human readable form, to a file.
  -- Exceptions may or may not be propagated.
  procedure Brief
    ( F : File_Type;
      Typ : Stat.Id;
      Var : VARIABLE );
  -- Writes Brief report to a file according to format
  -- specifications defined in package Stat. File contains
  -- statistics and is intended for graphing purposes.
  -- Type is a 4 character string ID provided by the user.

end Ai;
```

6.8 PACKAGE Af SPECIFICATION

This specification, given below, defines a variable type for maintaining statistical information and provides the procedures required for maintaining and reporting accumulated floating point statistics.

```
with Sim;
use Sim;
with Text_IO;
use Text_IO;
with Stat;
------------------------------------------------------------
package Af is
------------------------------------------------------------

  -- Variable is the type definition for records
  -- where statistical information is stored.
  -- The discriminant Seq is a sequence number that
  -- is intended as an ID number for graphing purposes.
  -- Procedure Brief writes statistics to a file
  -- which later may be graphed.
  type VARIABLE
    ( Seq : NATURAL := Stat.Seq(Stat.Af) ) is
    record
      Initialized : boolean := false;
      T_0 : Real := 0.0;
      T_Last : Real := 0.0;
      Sum_X : Real := 0.0;
```

continued

```
    Sum_Squares_X : Real := 0.0;
    Max_X : Real := Real'First;
    Min_X : Real := Real'Last;
    X : Real;
  end record;

-- Error_Type contains types of error that may be
-- propagated up from the subprograms in this
-- package.
type ERROR_TYPE is
  ( NONE, CONSTRAINT, NODATA, OTHER );

procedure Reset
  ( Var : in out VARIABLE );
-- Reset Stats.

procedure Assign
  ( Var : in out VARIABLE;
    Value : Real );
-- Assign a new value to this variable.

-- The equations below, which came from the text,
-- "SIMSCRIPT II.5 Programming Language", C.A.C.I.,
-- (pg 390), are used to calculate the mean and
-- variance.
--
--   SUM              summation of x * [TIME.V - TL]
--   SUM.OF.SQUARES   summation of [x**2] * [TIME.V - TL]
--   MEAN             SUM/[TIME.V - T0]
--   MEAN.SQUARE      SUM.OF.SQUARES/[TIME.V - T0]
--   VARIANCE         MEAN.SQUARE - MEAN**2
--
--     where TIME.V is simulation clock time,
--           TL is the simulated time at which an
--              accumulated variable was set to its
--              current value, and
--           T0 is the simulated time at which
--              accumulation starts

procedure Mean
  ( Var : VARIABLE;
    Result : out Real;
    Error : out ERROR_TYPE;
    Propagate : BOOLEAN := TRUE );
-- Return the average of the variable.
-- Exceptions may or may not be propagated.

procedure Variance
  ( Var : VARIABLE;
    Result : out Real;
    Error : out ERROR_TYPE;
    Propagate : BOOLEAN := TRUE );
-- Return the variance of the variable values.
-- Exceptions may of may not be propagated

procedure Std_Dev
  ( Var : VARIABLE;
    Result : out Real;
    Error : out ERROR_TYPE;
    Propagate : BOOLEAN := TRUE );
-- Return standard deviation of variable values.
-- Exceptions may or may not be propagated

procedure Report
```

continued

```
    ( Var : VARIABLE;
      Propagate : BOOLEAN := False );
  -- Prints statistics, in human readable form, to monitor.
  -- Exceptions may or may not be propagated.

  procedure Report
    ( F : File_Type;
      Var : VARIABLE;
      Propagate : BOOLEAN := False );
  -- Prints statistics, in human readable form, to a file.
  -- Exceptions may or may not be propagated.

  procedure Brief
    ( F : File_Type;
      Typ : Stat.Id;
      Var : VARIABLE );
  -- Writes Brief report to a file according to format
  -- specifications defined in package Stat. File contains
  -- statistics and is intended for graphing purposes.
  -- Typ is a 4 character string ID provided by the user.
end Af;
```

6.9 PACKAGE Ti, Tf, Ai, Af EXAMPLE

The example below shows how the subprograms in these four packages are called.

```
-- This procedure illustrates the use of the visible
-- subprograms contained in packages Ti, Tf, Ai, and Af.

with Stat;
with EV;
with SIM;
with TF;
with TI;
with AF;
with AI;
with Text_IO;
use SIM; -- Type Real is used from package SIM.
         -- Ev.Clock is of type SIM.Real
procedure Stat_Example is

  Type Test_Type is (TF_Test,TI_Test,AF_Test,AI_Test);
  package Enum_IO is new Text_IO.Enumeration_IO(Test_Type);
  -- Define parameters of variable type for the four
  -- packages.
  Tallied_Var_Tf : TF.VARIABLE;
  Tallied_Var_Ti : TI.VARIABLE;
  Accum_Var_AF : AF.VARIABLE;
  Accum_Var_AI : AI.VARIABLE;

  -- The result and error variables, which are defined
  -- below, return the results and error_type, respectively,
  -- from the Mean, Variance, and Std_Dev procedures.
  Result_Tf : Real;
  Error_Tf : TF.Error_Type;
  Result_Ti : Real;
  Error_Ti : TI.Error_Type;

                                                    continued
```

```
    Result_Af : Real;
    Error_Af : AF.Error_Type;
    Result_Ai : Real;
    Error_Ai : AI.Error_Type;
    -- Test_T is a parameter that indicates which package
    -- will be tested. Test_T will be input below.
    Test_T : Test_Type;
    -- Prop = true indicates that exceptions that occur in
    -- procedure Mean, Variance, Std_Dev, and Report will be
    -- propagated up.
    Prop : boolean := true;
    -- Otf and Otf2 are used when statistics are written to
    -- a file using procedure Report and Brief
    Otf : Text_IO.File_Type;
    Otf2 : Text_IO.File_Type;
    -- The IDs below are used in procedure Brief. They
    -- are used for graphing purposes to identify which
    -- call to Brief the statistics are associated with.
    ID : Stat.ID := "0001";
    ID2 : Stat.ID := "ND02";
    ID3 : Stat.ID := "ND03";
    ID4 : Stat.ID := "ND04";
  begin
    Text_IO.Create(Otf2,Text_IO.Out_File,"Test_Output");
    Text_IO.put
      ("enter TF_TEST, TI_TEST, AF_TEST, or AI_TEST >");
    Enum_IO.get(Test_T);
    case Test_T is
      when TF_TEST =>
        -- Result_Tf, which is returned from Mean, Variance,
        -- and Std_Dev below contains 0.0 and [no data] is
        -- printed to the users monitor from Report. This is
        -- true since no value has been assigned to
        -- Tallied_Var_Tf. These procedures are normally
        -- called only after a value has been assigned. If
        -- an error occurs, then it is propagated up.
        TF.Mean(Tallied_Var_Tf,Result_Tf,Error_Tf,Prop);
        TF.Variance(Tallied_Var_Tf,Result_Tf,Error_Tf,Prop);
        TF.Std_Dev(Tallied_Var_Tf,Result_Tf,Error_Tf,Prop);
        TF.Report(Tallied_Var_Tf,Prop);

        -- The values 0.0 and 1.0 are assigned. This means
        -- that statistics for these values are being
        -- maintained.
        TF.Assign(Tallied_Var_Tf,0.0);
        TF.Assign(Tallied_Var_Tf,1.0);
        -- The variance returned below in Result_Tf is 0.25
        -- If an error occurs, it is not propagated up.
        TF.Variance(Tallied_Var_Tf,Result_Tf,Error_Tf);
        Ev.Clock := 3.0;
        TF.Assign(Tallied_Var_Tf,2.0);

        -- Create file, TF_Test, that will contain statistics.
        Text_IO.Create(Otf,Text_IO.OUT_FILE,"TF_TEST");
        -- The file, TF_Test, contains statistics.
        TF.Brief(Otf,ID3,Tallied_Var_Tf);
        -- Report prints the following information to the
        -- user's monitor:
        -- X:2.000 Mean: 1.000 Std: 0.816 Var: 0.667
        -- Obs: 3 Sum:3.000 Min:0.000 Max:2.000
        TF.Report(Tallied_var_TF);
        EV.Clock := EV.Clock + 1.0;
        TF.Assign(Tallied_Var_Tf,3.0);
        EV.Clock := EV.Clock + 1.0;
```

continued

```
    TF.Brief(Otf,ID3,Tallied_Var_Tf);
    -- Report prints the following information to
    -- the user's monitor:
    -- X:4.00 Mean:2.000 Std: 1.414 Var: 2.000
    -- Obs: 5 Sum:10.000 Min:0.000 Max:4.000
    TF.Report(Tallied_Var_Tf);
    TF.Assign(Tallied_Var_Tf,4.0);
    TF.Mean(Tallied_Var_Tf,Result_Tf,Error_Tf);
    TF.Variance(Tallied_Var_Tf,Result_Tf,Error_Tf);
    TF.Std_Dev(Tallied_Var_Tf,Result_Tf,Error_Tf);
    TF.Report(Tallied_Var_Tf);
    -- Do report to file. If an error occurs do not
    -- propagate it up.
    Prop := false;
    TF.Report(Otf2,Tallied_Var_Tf,Prop);
    TF.Reset(Tallied_Var_Tf);
  when TI_TEST =>
    TI.Mean(Tallied_Var_TI,Result_TI,Error_TI,Prop);
    TI.Variance(Tallied_Var_TI,Result_TI,Error_TI,Prop);
    TI.Std_Dev(Tallied_Var_TI,Result_TI,Error_TI,Prop);
    -- The following information is printed to the user's
    -- monitor from report:
    -- [no data]
    TI.Report(Tallied_Var_TI,Prop;;

    TI.Assign(Tallied_Var_Ti,0);
    TI.Assign(Tallied_Var_Ti,1);
    TI.Variance(Tallied_Var_Ti,Result_Ti,Error_Ti);
    TI.Assign(Tallied_Var_Ti,2);
    EV.Clock := 3.0;

    -- Test TI.Brief

    Text_IO.Create(Otf,Text_IO.Out_File,"TI_Test");
    TI.Brief(Otf,ID4,Tallied_Var_TI);
    -- The following information is printed out
    -- from Report:
    -- X: 2 Mean: 1.000 Std: 0.816 Var 0.667
    -- Obs: 3 Sum 3 Min: 0 Max: 2
    TI.Report(Tallied_Var_TI);

    EV.Clock := EV.Clock + 1.0;
    TI.Assign(Tallied_Var_Ti,3);
    EV.Clock := EV.Clock + 1.0;
    TI.Brief(Otf,ID4,Tallied_Var_TI);
    -- The following information is printed out
    -- from Report:
    -- X: 3 Mean: 1.500 Std: 1.118 Var: 1.250
    -- Obs: 4 Sum: 6 Max: 3
    TI.Report(Tallied_Var_TI);

    TI.Assign(Tallied_Var_Ti,4);
    TI.Mean(Tallied_Var_Ti,Result_Ti,Error_Ti);
    TI.Variance(Tallied_Var_Ti,Result_Ti,Error_Ti);
    TI.Std_Dev(Tallied_Var_Ti,Result_Ti,Error_Ti);
    -- The following information is printed out
    -- from Report:
    -- X: 4 Mean: 2.000 Std: 1.414 Var: 2.000
    -- Obs: 5 Sum: 10 Min: 0 Max: 4
    TI.Report(Tallied_Var_Ti);
    -- Do report to file.
    Prop := false;
    -- Same information as above is written to
    -- file test_output.
```

continued

```
     TI.Report(Otf2,Tallied_Var_Ti,Prop);
     -- Reset resets the statistics, as though no
     -- values have been collected.
     TI.Reset(Tallied_Var_Ti);
   when AF_TEST =>
     AF.Mean(Accum_Var_AF,Result_AF,Error_AF,Prop);
     AF.Variance(Accum_Var_AF,Result_AF,Error_AF,Prop);
     AF.Std_Dev(Accum_Var_AF,Result_AF,Error_AF,Prop);
     -- Report prints to the users monitor: [no data]
     AF.Report(Accum_Var_AF,Prop);

     EV.Clock := 1.0;

     AF.Assign(Accum_Var_AF,0.0);
     Ev.Clock := Ev.Clock + 1.0;
     AF.Assign(Accum_Var_AF,1.0);
     AF.Variance(Accum_Var_AF,Result_Af,Error_Af);
     EV.Clock := Ev.Clock + 1.0;
     AF.Assign(Accum_Var_AF,2.0);

     -- Test AF.Brief

     Text_IO.Create(Otf,Text_IO.out_file,"AF_Test");
     AF.Brief(Otf,ID2,Accum_Var_AF);
     -- Report prints the following to the user's
     -- monitor:
     -- X:000 Mean: 0.500 Std: 0.500 Var: 0.250
     -- Sum: 1.000 Min: 0.000 Max: 2.000
     AF.Report(Accum_var_AF);

     EV.Clock := Ev.Clock + 1.0;
     AF.Assign(Accum_Var_AF,3.0);
     EV.Clock := Ev.Clock + 1.0;
     AF.Brief(Otf,ID2,Accum_Var_AF);
     AF.Report(Accum_var_AF);

     AF.Assign(Accum_Var_AF,4.0);
     AF.Mean(Accum_Var_AF,Result_Af,Error_Af);
     AF.Variance(Accum_Var_AF,Result_Af,Error_Af);
     AF.Std_Dev(Accum_Var_AF,Result_Af,Error_Af);
     -- Report writes out the following:
     -- X:3.00 Mean: 1.500 Std: 1.118 Var: 1.250
     -- Sum : 3.000 Min: 0.000 Max 3.000
     AF.Report(Accum_Var_AF);
     Prop := false;
     AF.Report(Otf2,Accum_Var_Af,Prop);
     -- Do report to file.
     AF.Reset(Accum_Var_AF);
   when AI_Test =>
     AI.Mean(Accum_Var_AI,Result_AI,Error_AI,Prop);
     AI.Variance(Accum_Var_AI,Result_AI,Error_AI,Prop);
     AI.Std_Dev(Accum_Var_AI,Result_AI,Error_AI,Prop);

     -- Report prints out : [no data]
     AI.Report(Accum_Var_AI,Prop);

     EV.Clock := 1.0;
     AI.Reset(Accum_Var_AI);
     AI.Assign(Accum_Var_AI,0);
     Ev.Clock := Ev.Clock + 1.0;
     AI.Assign(Accum_Var_AI,1);
     AI.Variance(Accum_Var_AI,Result_AI,Error_AI);
     EV.Clock := Ev.Clock + 1.0;
     AI.Assign(Accum_Var_AI,2);
```

continued

```
      -- Try AI.Brief

      Text_IO.Create(Otf,Text_IO.Out_File,"AI_Test");
      EV.CLOCK := EV.CLOCK + 1.0;
      AI.Brief(Otf,ID,Accum_Var_AI);
      -- Report prints out the following:
      -- x: 2 Mean 1.000 Std: 0.816 Var: 0.667
      -- Sum: 1.000 Min: 0 Max: 2
      AI.Report(Accum_var_AI);

      EV.Clock := Ev.Clock + 1.0;
      AI.Assign(Accum_Var_AI,3);
      EV.Clock := Ev.Clock + 1.0;
      AI.Brief(Otf,ID,Accum_Var_AI);
      -- Report prints out the following:
      -- x: 3 Mean: 1.600 Std: 1.020 Var: 1.040
      -- Sum: 5.000 Min: 0 Max: 3
      AI.Report(Accum_var_AI);
      AI.Assign(Accum_Var_AI,4);
      AI.Mean(Accum_Var_AI,Result_AI,Error_AI);
      AI.Variance(Accum_Var_AI,Result_AI,Error_AI);
      AI.Std_Dev(Accum_Var_AI,Result_AI,Error_AI);
      -- AI.Report(Accum_Var_AI);
      -- Do report to file.
      Prop := false;
      AI.Report(Otf2,Accum_Var_AI,Prop);
      AI.Reset(Accum_Var_AI);
    end case;
end Stat_Example;
```

7.0 HASH TABLES

Evada contains two hash table packages. Hash tables provide a capability for quickly inserting, accessing, and deleting numbers in an array. The use of hash tables is appropriate when (1) these numbers have a wide range of values, and (2) there are relatively few numbers. A detailed explanation of hash tables is found in a book by Tennenbaum and Augenstein.* In an EARS simulation, hash tables may be used, for example, in duplicate message detection.

7.1 PACKAGE Hsh

Package Hsh, whose specification is given below, contains subprograms for creating, accessing, and deleting hash tables. The comments within the specification explain what these subprograms do.

```
with Nat_Lnk, Unchecked_Deallocation;
use Nat_Lnk;

package Hsh is

  Undef_Mem_Loc, Passed_End_Of_Table, Corrupted_Hash_Table
    : exception;

  Redundant_Inclusion
    : exception;

  type TABLE is
    private;

  -- Init creates a table whose size is the smallest
  -- prime number greater than Size.
  function Init
    ( Size : POSITIVE )
      return TABLE;

  -- Include inserts the value Nat into table Ht.
  procedure Include
    ( Ht : TABLE;
      Nat : NATURAL );

  -- Included tells whether or not Nat is a member
  -- of table Ht.
  function Included
    ( Ht : TABLE;
      Nat : NATURAL )
      return BOOLEAN;

  -- Members returns the number of members in table
  -- Ht.
  function Members
    ( Ht : TABLE )
      return NATURAL;

  -- Remove removes a member Nat from table Ht.
  procedure Remove
    ( Ht : TABLE;
      Nat : NATURAL;
      Found : in out BOOLEAN );
```

continued

*Tennenbaum, A. and M. Augenstein. 1981. *Data Structures Using Pascal.* Prentice-Hall, Englewood Cliffs, NJ.

```
  -- Traverse returns the first member of the
  -- table Ht in Nat. When the table has
  -- no members then Found is false.
  procedure Traverse
    ( Ht : TABLE;
      Nat : in out NATURAL;
      Found : in out BOOLEAN );

  -- Next may only be called after Traverse Next,
  -- or delete. For non-empty tables returns
  -- member Nat. For empty tables returns Found
  -- set to false.
  procedure Next
    ( Ht : TABLE;
      Nat : in out NATURAL;
      Found : in out BOOLEAN );

  -- Delete may only be called after Traverse Next,
  -- or delete. For non-empty tables returns
  -- member Nat and deletes it from table Ht.
  procedure Delete
    ( Ht : TABLE;
      Nat : in out NATURAL;
      Found : in out BOOLEAN );

  -- Dump removes all members from the table.
  procedure Dump
    ( Ht : in out TABLE );

  -- Destroy removes all members from the
  -- table and deallocates the memory.
  procedure Destroy
    ( Ht : in out TABLE );

  -- Package Sort sorts the table into a
  -- linked list according to the criteria
  -- defined in the user specified function.
  generic
    with function ">"
      ( M1, M2 : NATURAL )
        return BOOLEAN;
  package Sort is
    type NATURAL_STRUCTURE is
      array (NATURAL range <>)
        of NATURAL;

    type NATURAL_ARRAY is
      access NATURAL_STRUCTURE;

    -- Produce produces a sorted array of elements.
    function Produce
      ( Ht : TABLE )
        return NATURAL_ARRAY;

    -- Free deallocates memory used by array.
    procedure Free is
      new Unchecked_Deallocation
      (NATURAL_STRUCTURE,
    NATURAL_ARRAY);

  end Sort;

-- Private portion of specification deleted in
-- this documentation

end Hsh;
```

7.2 PACKAGE Hash

Package Hash, whose specification appears below, provides capabilities that are similar to those provided by Hsh. Package Hash is a generic package that is instantiated with the type REC, its access type, MEMBER, and two user-provided functions. The first function, Same, compares two members, while the second, Key, calculates a unique integer key for a member.

The package specification for Hash, with comments describing what the subprograms do, is given below.

```
with Unchecked_Deallocation, Link;

generic

  type REC is
    private;

  type MEMBER is
    access REC;

  -- The user writes a function Same that compares two
  -- members and returns true when they match.
  with function Same
    ( M1, M2 : MEMBER )
      return BOOLEAN;

  -- The user writes a function that returns an integer
  -- key that is unique to the member.
  -- The key is used by the Hash subprograms to access
  -- the member.
  -- An explanation of the use of keys is found in
  -- data structures texts, such as, "Data Structures
  -- Using Pascal", by Tenenbaum and Augenstein.
  with function Key
    ( Mbr : MEMBER )
      return INTEGER;

package Hash is

  Undef_Mem_Loc, Passed_End_Of_Table, Corrupted_Hash_Table
    : exception;

  Redundant_Inclusion
    : exception;

  package Lnk is
    new Link (REC, MEMBER);

  use Lnk;

  type TABLE is
    private;

  -- Init creates a table whose Size is the smallest
  -- prime number greater than size.
  function Init
    ( Size : NATURAL )
      return TABLE;

  -- Include inserts the value Nat into table Ht.
  procedure Include
```

continued

```
    ( Ht : TABLE;
      Mbr : MEMBER );

  -- Included tells whether or not Nat is a member
  -- of table Ht.
  function Included
    ( Ht : TABLE;
      Mbr : MEMBER )
      return BOOLEAN;

  -- Members returns the number of members in table
  -- Ht.
  function Members
    ( Ht : TABLE )
      return NATURAL;

  -- Find searches the list to find a list member
  -- such that Same(Match,Mbr) is true.
  procedure Find
    ( Ht : TABLE;
      Match : MEMBER;
      Mbr : out MEMBER;
      Found : in out BOOLEAN );

  -- Remove removes a member Nat from table Ht.
  procedure Remove
    ( Ht : TABLE;
      Match : MEMBER;
      Mbr : out MEMBER;
      Found : in out BOOLEAN );

  -- Traverse returns the first member of the
  -- table Ht in Mbr. When the table has no
  -- members, then Found is false.
  procedure Traverse
    ( Ht : TABLE;
      Mbr : in out MEMBER;
      Found : in out BOOLEAN );

  -- Next may only be called after Traverse, Next
  -- or delete. For non-empty tables it returns
  -- member Nat. For empty tables it returns Found
  -- set to false.
  procedure Next
    ( Ht : TABLE;
      Mbr : in out MEMBER;
      Found : in out BOOLEAN );

  -- Delete may only be called after Traverse,
  -- Next, or Delete. For non-empty tables it
  -- returns member Mbr and deletes it from
  -- table Ht.
  procedure Delete
    ( Ht : TABLE;
      Mbr : in out MEMBER;
      Found : in out BOOLEAN );

  -- Dump removes all members from the table.
  procedure Dump
    ( Ht : in out TABLE );

  -- Destroy removes all members from the
```

continued

```
  -- table and deallocates the memory.
  procedure Destroy
    ( Ht : in out TABLE );

  generic
    with function ">"
      ( M1, M2 : MEMBER )
        return BOOLEAN;

  -- Produce produces a sorted array of
  -- elements.
  package Sort is
    type MEMBER_STRUCTURE is
      array (NATURAL range <>)
        of MEMBER;

    type MEMBER_ARRAY is
      access MEMBER_STRUCTURE;

    -- Produce produces a sorted array of elements.
    function Produce
      ( Ht : TABLE )
        return MEMBER_ARRAY;

    -- Free deallocates memory used by array.
    procedure Free is
      new Unchecked_Deallocation
      (MEMBER_STRUCTURE, MEMBER_ARRAY);

  end Sort;

private -- Hash

  type Lst is
    access List;

  type VAR is
    record
      Chained : BOOLEAN := False;
      Chain : Lst := null;
      Mbr : MEMBER := null;
    end record;

  type HASH_ARRAY is
    array (NATURAL range <>)
      of VAR;

  type CTRL_REC is
    record
      Current : NATURAL := 0;
      Inlist : BOOLEAN := False;
      Eot : BOOLEAN := TRUE;
    end record;

  type TABLE_ARRAY
    ( Size : NATURAL ) is
    record
      Store : HASH_ARRAY (0 .. Size);
      Ctrl : CTRL_REC;
      Members : NATURAL := 0;
    end record;

  type TABLE is
    access TABLE_ARRAY;

end Hash;
```

8.0 BIBLIOGRAPHY

Booch, G. 1987. *Software Engineering with Ada*. Benjamin/Cummings Publishing Company, Menlo Park, CA.

Russell, E. 1983. *Building Simulation Models*. C.A.C.I., Inc., Los Angeles, CA.

Graybeal, W., and U. Pooch. 1980. *Simulation Principles and Methods*. Winthrop Publishers, Cambridge, MA.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1990	3. REPORT TYPE AND DATES COVERED November 1989 – July 1990

4. TITLE AND SUBTITLE USER'S GUIDE TO AN EVENT-ACTIVATION RECORD APPROACH TO SIMULATION MODELING IN ADA	5. FUNDING NUMBERS PE: 0602232N PROJ: RC32A13 WU: DN309082
6. AUTHOR(S) H. Mumm and R. Ollerton	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Ocean Systems Center San Diego, CA 92152–5000	8. PERFORMING ORGANIZATION REPORT NUMBER NOSC TD 1944
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Naval Ocean Systems Center Block Programs San Diego, CA 92152-5000	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This document explains how to install and use the Event-Activation Record Approach to Simulation Modeling in Ada (EARS) simulation libraries, Version 1.0, that were developed by the Naval Ocean Systems Center, Code 854. This software was designed to compile and execute using the Vax Ada compiler. EARS is a discrete-event simulation system that uses an event-activation record approach. It has capabilities similar to those provided by SIMSCRIPT. All simulations that can be modeled in SIMSCRIPT can be modeled using EARS.

From a set of user-defined event types, EARS automatically generates much of the source code for many Ada packages and subprograms that are needed for simulation development. This allows a user to quickly start writing an EARS simulation. The user tailors the simulation to a particular application by adding Ada source code to the generated program units and by writing additional ones.

This document contains an actual simulation example using EARS. The document also includes the Ada source code for linked list packages, statistical packages, and hash table packages, as well as examples of how they are used.

14. SUBJECT TERMS Ada instantiation elaboration	EARS linked list discrete-event simulation hash tables	15. NUMBER OF PAGES 77
		16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

INITIAL DISTRIBUTION

CODE 0012	Patent Counsel	(1)
CODE 0144	R. November	(1)
CODE 171	Dr. M. Vineberg	(1)
CODE 40	Dr. R. C. Kolb	(1)
CODE 41	A. Justice	(1)
CODE 411	D. Hayward	(1)
CODE 411	B. Ollerton	(1)
CODE 411	H. Mumm	(8)
CODE 85	R. Casey	(1)
CODE 8503	Dr. C. Warner	(1)
CODE 854	R. Merk	(1)
CODE 854	Dr. N. Dave'	(1)
CODE 921	J. Puleo	(1)
CODE 961	Archive/Stock	(6)
CODE 964	Library	(3)

Defense Technical Information Center Alexandria, VA 22304-6145	(4)
NOSC Liaison Office Washington, DC 20363-5100	(1)
Center for Naval Analyses Alexandria, VA 22302-0268	(1)