Prototype Real-Time Monitor:
Ada Code
Roger Van Scoy
November 1987
Prototype Real-Time Monitor: Ada Code

Roger Van Scoy

Dissemination of Ada Software Engineering Technology

Approved for public release.
Distribution unlimited.

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
# Table of Contents

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>package Test_Stub</td>
<td>2</td>
</tr>
<tr>
<td>package body Test_Stub</td>
<td>2</td>
</tr>
<tr>
<td>package Real_Time_Monitor</td>
<td>4</td>
</tr>
<tr>
<td>package body Real_Time_Monitor</td>
<td>6</td>
</tr>
<tr>
<td>package Rtm_Form</td>
<td>14</td>
</tr>
<tr>
<td>package body Rtm_Form</td>
<td>18</td>
</tr>
<tr>
<td>package Define_Rtm_Cli</td>
<td>27</td>
</tr>
<tr>
<td>package Rtm_Cli</td>
<td>27</td>
</tr>
<tr>
<td>package body Define_Rtm_Cli</td>
<td>30</td>
</tr>
<tr>
<td>package SA</td>
<td>30</td>
</tr>
<tr>
<td>package SI</td>
<td>30</td>
</tr>
<tr>
<td>package SP</td>
<td>33</td>
</tr>
<tr>
<td>package Page_Processor</td>
<td>35</td>
</tr>
<tr>
<td>package body Page_Processor</td>
<td>41</td>
</tr>
<tr>
<td>package Duration Io</td>
<td>41</td>
</tr>
<tr>
<td>package Field_Lists</td>
<td>42</td>
</tr>
<tr>
<td>package CLI</td>
<td>45</td>
</tr>
<tr>
<td>package CLI</td>
<td>47</td>
</tr>
<tr>
<td>package CiISC</td>
<td>49</td>
</tr>
<tr>
<td>package CLI</td>
<td>49</td>
</tr>
<tr>
<td>package CLI</td>
<td>52</td>
</tr>
<tr>
<td>package CLI</td>
<td>57</td>
</tr>
<tr>
<td>package Parameter_Manager</td>
<td>61</td>
</tr>
<tr>
<td>package body Parameter_Manager</td>
<td>64</td>
</tr>
<tr>
<td>package CLI</td>
<td>65</td>
</tr>
<tr>
<td>package CLI</td>
<td>67</td>
</tr>
</tbody>
</table>

October 1987
package Conversions 71
generic package Convert_Integers 73
generic package Convert_Floats 75
generic package Convert_Enumerations 77
package body Conversions 79
package body Convert_Integers 79
package body Convert_Floats 79
package body Convert_Enumerations 79
package body Convert_Integers 81
package Internal_lo 81
package body Convert_Floats 85
package Internal_lo 85
package body Convert_Enumerations 89
package Internal_lo 89
package Types_Manager 93
package body Types_Manager 96
package Rtm_Reals 98
package Rtm_Integers 99
package Rtm.Enums 100
package CISC 104
package Variable_Database 108
package body Variable_Database 110
package CISC 110
package DB 111
package Library_Interface 117
package body Library_Interface 120
package Address_Generator 126
package body Address_Generator 127
package Dialogue_Manager 130
package body Dialogue_Manager 139
package Collect_Data 139
package Active_Lists 140
package body Collect_Data 140
package body Collect_Data 149
package Sysgen 156
package Rtm_Core 159
package body Rtm_Core 163
Index 171
with Text_io; use Text_io;
with Test_Stub;
with Real_Time_Monitor;

procedure Appl is

begin
  loop
    Test_Stub.Go;
    begin
      Real_Time_Monitor.Rtm;
      exception
      when Real_Time_Monitor.Terminate_Rtm =>
        Put_Line("RTM terminated, application still running");
    end;
  end loop;
end Appl;
pragma page;
package Test_Stub is
  type Rtm_Record is record
    I: Integer;
    R: Float;
  end record;
  type Rtm_Pointer is access Rtm_Record;
  type Rtm_Enum is (Hehe, Haha, Hoho);

  My_Array: array (1..5) of Integer := (1,2,3,4,5);
  My_Pointer: Rtm_Pointer := new Rtm_Record'I => 2, R => 1000.0);
  My_Integer: Integer := 2;
  Int_2: Integer := 337;
  My_Real: Float := 10.0;
  My_Enum: Rtm_Enum := Hehe;

  procedure Go;
end Test_Stub;

package Test_Stub is
  procedure Go is
    begin
      My_Pointer.I := My_Pointer.I + 2;
      My_Real := My_Real + 1.0;
      My_Integer := My_Integer + 1;
    end Go;
end Test_Stub;
pragma page;
Module Name:
Real_Time_Monitor

Module Type:
Package Specification

Module Purpose:
This package is the main driver for the RTM.

Module Description:
Implements the real-time monitor abstraction, i.e., all the
commands found in the RTM User's Manual.

References:
Design Documents:
Real-Time Monitor Requirements
Real-Time Monitor Design

User's Manual:
RTM User's Manual

Testing and Validation:
none

Notes:
none

Modification History:
15Apr87 rvs Created

Distribution and Copyright Notice:
TBD

Disclaimer:
"This work was sponsored by the Department of Defense,
The views and conclusions contained in this document are
solely those of the author(s) and should not be interpreted as
representing official policies, either expressed or implied,
of Carnegie Mellon University, the U.S. Air Force,
or the Department of Defense, or the U.S. Government."

pragma page;
package Real_Time_Monitor is
  ...
  -- Signals to the controlling program the termination of the real-time
  -- monitor.
  ...
  Terminate_Rtm: exception;
  
  procedure Rtm;              
  --|--------------------------------------------------------------------------------
  --| Description:
  --| This module is the RTM proper. See the documents associated
  --| with the RTM for a complete description of what it does.
  --|
  --| Parameter Description:
  --| none
  --|--------------------------------------------------------------------------------
  
end Real_Time_Monitor;
pragma page;
Module Name:

RealTimeMonitor

Module Type:

Package Body

Module Description:

This package just ties together the services needed to execute the RTM:

- the initialization procedure
- the command processor procedure
- the termination procedure
- and the RTM itself

References:

Design Documents:
- Real-Time Monitor Requirements
- Real-Time Monitor Design

User's Manual:
- RTM User's Manual

Testing and Validation:

- none

Notes:

- none

Modification History:

15Apr87 rvs Created

Distribution and Copyright Notice:

TBD

Disclaimer:

"This work was sponsored by the Department of Defense. The views and conclusions contained in this document are solely those of the author(s) and should not be interpreted as representing official policies, either expressed or implied, of Carnegie Mellon University, the U.S. Air Force, the Department of Defense, or the U.S. Government."

pragma page;
with Rtm_Form; use Rtm_Form;
-- Use all the services.
--
with Define_Rtm_Cli; use Define_Rtm_Cli;
-- Use the type "rtm_command_representation".
-- Use all the services.
--
with Terminal_Interface;
-- Use the "open" and "close" services.
--

package Real_Time_Monitor is
--
-- Define the structures needed by the RTM to interface to the user.
--
User_Command_Line: String (1..80);
User_Command_Length: Natural;
User_Line_Ready: Boolean := False;
Command_Found: Rtm_Command_Representation;
--
-- Internal Procedures
--
procedure Setup_Rtm is separate;
procedure Closeout_Rtm is separate;
procedure Process_The_Command (Users_Command : In Rtm_Command_Representation)
  is separate;
--
-- Visible Procedures
--
procedure Rtm is separate;
--
begin
  Setup_Rtm;
end Real_Time_Monitor;
pragma page;
with Standard_Interface;

-- Use the exceptions "abort_process", "undefined_name", "no_default",
-- "abort_command" and "no_command".

separate (Real_Time_Monitor)
procedure Rtm is

/**************************************************************/

--/ Description:
--/ A complete description of the RTM and its functioning can
--/ be found in the documents referenced in the spec. Here we
--/ will just describe how the RTM implements those functions
--/ at the highest level. The RTM is organized as a cyclic executive:
--/ 1. It polls the user for input.
--/ 2. If the input is ready,
--/   a. parse the command
--/   b. execute the command (which happens only if a legal
--/      command was found).
--/ 3. Update any active pages. This happens every cycle through
--/    the RTM, regardless of whether the user types a command
--/    (legal or otherwise).
--/

--; Parameter Description:
--; none
--; /
--; Notes:
--; none
--; /

begin

begin

-- Attempt to get the user's command.

--

Get_Rtm_Field (Field => Rtm_Command,
Field_Value => User_Command_Line,
Data_Available => User_Line_Ready);

-- When a command has been found,
-- - Parse the command
-- - Process the command
-- - Reset the parser for the next command line
--

If User_Line_Ready then

Command_Found := Rtm_Cli.Parse_Command_Line
(Rtm_Commands,User_Command_Line);

Process_The_Command (Users_Command => Command_Found);

Clear_Command_Line (Command_Found);

end if;

exception

-- The parser has a large number of exceptions that signal a bad
-- command; all of them are handled here, and relayed to the user.

when Standard_Interface.Abort_Process |
Standard_Interface.Undefiend_Name |
Standard_Interface.No_Default |
Standard_Interface.Abort_Command =>
    Put_Rtm_Field (Field => Message_Field_A,
                   Field_Value => User_Command_Line);
    Put_Rtm_Field (Field => Message_Field_B,
                   Field_Value => "Bad commad line, reenter");
when Standard_Interface.No_Command =>
    null ;
when Terminate_Rtm =>
    RAISE ;
when others =>
    Put_Rtm_Field (Field => Message_Field_B,
                   Field_Value => "Bad Command line");
end ;

-- Finally, after the user command has been processed, we
-- perform our periodic update of any active pages
--
    Process_The_Command (Users_Command => Update_Active_Pages);
    Clear_Rtm_Field;
end Rtm;
pragma page;
procedure Setup_Rtm is

-- Description:
-- This module is responsible for performing all initialization
-- required by the RTM prior to execution. It must:
-- 1. Open a terminal channel for I/O.
-- 2. Create the RTM forms.
--
-- Parameter Description:
-- none
--
-- Notes:
-- none
--
begin
  Terminal_Interface.Open;
  Initialize_Rtm_Form;
end Setup_Rtm;

pragma page;
separate (Real_Time_Monitor)

procedure Closeout_Rtm is

-- | Description:
-- | This module performs all termination operations needed after
-- | completion of the RTM. It must:
-- | Close the terminal I/O channel.
-- |

-- | Parameter Description:
-- | none
-- |

-- | Notes:
-- | none
-- |

begin
  Terminal_Interface.Close;
end Closeout_Rtm;

pragma page;
with Interact;
-- Use the service "interact".
-
with Terminal_Interface;
-- Use the service "clear_screen".
-
with Page_Processor;
-- Use the services "start_page", "stop_page", "check_page" and
-- "update_pages".
-
with Parameter_Manager;
-- Use the services "read" and "set".
-
separate (Real_Time_Monitor)
procedure Process_The_Command (Users_Command: in Rtm_Command_Representation) is
  -- Description:
  -- This command invokes the modules which implement the various
  -- commands.
  --
  -- Parameter Description:
  -- users_command -> The command identifier for the most recently,
  -- successfully parsed user command.
  --
  -- Notes:
  -- The Quit(); command is implemented in this module by raising
  -- the Terminate_Rtm exception, which is propagated out.
  --
begin
  case Users_Command is
    when Edit =>
      Interact;
      Terminal_Interface.Clear_Screen;
    when Quit =>
      RAISE Terminate_Rtm;
    when Read =>
      Parameter_Manager.Read;
    when Set =>
      Parameter_Manager.Set;
    when Check =>
      Page_Processor.Check_Page;
    when Start =>
      Page_Processor.Start_Page;
    when Stop =>
      Page_Processor.Stop_Page;
    when Update_Active_Pages =>
      Page_Processor.Update_Pages;
    when others =>
      null;
  end case;
exception
when Terminate_Rtm =>
  Closeout_Rtm;
  RAISE;
when others =>
  null;

end Process_The_Command;
pragma page;
--/ Module Name:
--/ RTM_Form

--/ Module Type:
--/ Package Specification

--/ Module Purpose:
--/ This package abstracts away the details of using the
--/ forms management system for input and error messages.

--/ Module Description:
--/ This package hides from the RTM itself the actual details
--/ of dealing with the forms manager for I/O. The
--/ services provided include:
--/ 1. Initializing the message and prompt forms.
--/ 2. Presenting error messages to the user.
--/ 3. Setting up and retrieving data entered by the user
--/ into the prompt form.

--/ References:
--/ Design Documents:
--/ none

--/ User's Manual:
--/ none

--/ Testing and Validation:
--/ none

--/ Notes:
--/ none

--/ Modification History:
--/ 16Apr87 rfvs created

--/ Distribution and Copyright Notice:
--/ TBD

--/ Disclaimer:
--/ "This work was sponsored by the Department of Defense.
--/ The views and conclusions contained in this document are
--/ solely those of the author(s) and should not be interpreted as
--/ representing official policies, either expressed or implied,
--/ of Carnegie Mellon University, the U.S. Air Force,
--/ the Department of Defense, or the U.S. Government."

pragma page;
package Rtm_Form is

-- This type defines the different interface modes used to communicate
-- with the user. See package body for a complete description.

type Input_Mode is (Screen_Mode, Command_Mode);

-- This type defines the different fields available to the RTM on the
-- RTM interface form.
-- message_field_a is an output-only message field.
-- message_field_b is an output-only message field
-- rtm_command is an input-only field used only for obtaining
-- user inputs.
-- form is not a message field, but affects the rtm interface
-- form as a whole. This is a shorthand notation for
-- performing the same operation on all the message fields.

type Rtm_Form_Fields is (Message_Field_A, Message_Field_B, Rtm_Command, Form);

procedure Initialize_Rtm_Form;

-- /Description:
-- / Creates the RTM interface form and makes it available for use.
-- /
-- / Parameter Description:
-- / none
-- /
procedure Get_Rtm_Field (Field: In Rtm_Form_Fields;
 Field_Value: In out String;
 Data_Available: In out Boolean);

-- /Description:
-- / Displays the RTM prompt and retrieves data entered by the
-- / user in the form.
-- /
-- / Parameter Description:
-- / field -> The name of the field on the RTM interface form from
-- / which to retrieve the data.
-- / field_value -> The data entered by the user into the field.
-- / data_available -> A logical flag which indicates when data are
-- / available in the indicated field. Important
-- / when asynchronous I/O is being used to interface
-- / to the user's terminal.
-- /
procedure Put_Rtm_Field (Field: In Rtm_Form_Fields;
 Field_Value: In String := "");

-- /Description:
-- / Modifies the value of a field on the RTM interface form.

14 October 1987
It then presents the information to the user.

Parameter Description:
- field -> The name of the field on the RTM interface form in which to store the data.
- field_value -> The data to be stored in the field.

procedure Clear_Rtm_Field (Field: In Rtm_Form_Fields := Form);

Description:
Blanks out the current value of a field on the RTM interface form.

Parameter Description:
- field -> The name of the field on the RTM interface form to blank.

procedure Set_Input_Mode (Next_Mode: In Input_Mode := Command_Mode);

Description:
Select the next input mode for the RTM interface form.

Parameter Description:
- Next_mode -> Needed interface mode to the user.

end Rtm_Form;
pragma page;
Module Name:

IRTMForm

Module Type:

Package Body

Module Description:

This package implements the RTM interface form. This form is visible to the rest of the RTM as four fields available for I/O - in reality, it is implemented as two separate forms. The fields available to RTM are:

- Message_field_a: an 80 character output only field belonging to rtm_message_form.
- Message_field_b: an 80 character output only field belonging to rtm_message_form.
- Rtm_command: a 70 character input only field belonging to the rtm_prompt_form.
- Form: operates on all the fields above.

The input_mode is used to select either:

- screen_mode: a semi-asynchronous input, that is, when the screen is being updated rapidly, and the RTM shouldn't wait for input from the user, an asynchronous input is requested, and the RTM continues processing. When the user strikes any key, the rtm_prompt_form is displayed and the user enters data synchronously.
- command_mode: a simple synchronous input using the rtm_prompt_form.

References:

Design Documents:

none

Testing and Validation:

none

Notes:

This package makes use of SYSDEP, which contains all VAX/VMS dependent features used by the RTM.

Modification History:

16Apr87 rvs created

Distribution and Copyright Notice:

TBD

Disclaimer:

This work was sponsored by the Department of Defense. The views and conclusions contained in this document are solely those of the author(s) and should not be interpreted as...
with Sysdep;
-- Use the service that allows for asynchronous input of a character
-- from the terminal.
--
with Form_Manager;
-- Use services needed to create a form.
--
with Form_Executor;
-- Use the services needed to manipulate the forms of the user interface:
-- present_form -> to display a form to the user
-- modify_field -> to replace the value in a field of a form
-- query_field -> to retrieve the value in a field of a form
--

package Rtm_Form Is
--
-- The current input mode from the user, as described above.
--
Current_Input_Mode: Input_Mode := Command_Mode;
--
-- The form identifiers needed to access and manipulate the
-- forms of the user interface.
--
Rtm_Message_Form: Form_Manager.Form_Access;
Rtm_Prompt_Form: Form_Manager.Form_Access;

pragma page;
procedure Initialize_Rtm_Form is

--/ Description:
--/ Creates the RTM interface forms and makes them available for use
--/ internally. It does this by creating two forms and then
--/ defining their fields (one at a time). We do the creation here
--/ to avoid any disk dependencies and the chance that the user
--/ would modify the forms.
--/
--/ Parameter Description:
--/ none
--/
--/ Notes:
--/ none
--/........................................................................

  Field : Form_Manager.Field_Access;
begin
  --
  -- Create the output-only message form.
  --
  Form_Manager.Create_Form
  ((2, 80), (21, 1), Form_Manager.No_Clear, Rtm_Message_Form);
  Form_Manager.Add_Field
  (Rtm_Message_Form, "message-a", (1, 1), 80, Init_Value => "", 
   Mode => Form_Manager.Output_Only, Field => Field);
  Form_Manager.Add_Field
  (Rtm_Message_Form, "message-b", (2, 1), 80, Init_Value => "", 
   Mode => Form_Manager.Output_Only, Field => Field);

  -- Create the input-only prompt form.
  --
  Form_Manager.Create_Form
  ((2, 80), (23, 1), Form_Manager.No_Clear, Rtm_Prompt_Form);
  Form_Manager.Add_Field
  (Rtm_Prompt_Form, "message-a", (2, 1), 5, Init_Value => "rtm", 
   Mode => Form_Manager.Constant_Text, Field => Field);
  Form_Manager.Add_Field
  (Rtm_Prompt_Form, "message-b", (2, 6), 70, Init_Value => "", 
   Mode => Form_Manager.Input_Output, Field => Field);
exception
  when others =>
    null;
end Initialize_Rtm_Form;

pragma page;

19 October 1987
procedure Get_Rtm_Field (Field: in Rtm_Form_Fields;
    Field_Value: in out String;
    Data_Available: in out Boolean) is

-- Description:
-- Displays the RTM prompt and retrieves data entered in the form
-- by the user. If input is requested from one of the output
-- only fields or from the form as a whole, the data available
-- flag is returned as false (since no data can ever be obtained
-- from these fields). When input is requested from the rtm_prompt
-- field, where all the user input comes from, two situations exist:
-- in Screen mode: we check the terminal to see if a character
-- has been typed.
-- if not, data available is returned as false.
-- if so, then we place the rtm_prompt_form on the
-- terminal and prompt the user.
-- in Command mode: we place the rtm_prompt_form on the
-- terminal and prompt the user.

-- Parameter Description:
-- field -> The name of the field on the RTM interface form from
-- which to retrieve the data.
-- field_value -> The data entered by the user into the field.
-- data_available -> A logical flag which indicates when data are
-- available in the indicated field. Important
-- when asynchronous I/O is being used to interface
-- to the user's terminal.

-- Notes:
-- none

begin
    case Field is
        when Message_Field_A =>
            Data_Available := False;
        when Message_Field_B =>
            Data_Available := False;
        when Rtm_Command =>
            case Current_Input_Mode is
                when Screen_Mode =>
                    Sysdep.Get(Data_Available);
                    If Data_Available then
                        Form_Executor.Present_Form(Rtm_Prompt_Form);
                        Form_Executor.Query_Field(Form => Rtm_Prompt_Form,
                            Field => "rtm_command",
                            Value => Field_Value);
                    end If;
                when Command_Mode =>
                    Form_Executor.Present_Form(Rtm_Prompt_Form);
                    Form_Executor.Query_Field(Form => Rtm_Prompt_Form,
                        Field => "rtm_command",
                        Value => Field_Value);
                    Data_Available := True;
                end case;
        when Form =>

20 October 1987
Data_Available := False;
end case;

exception
  when others =>
    null;
end Get_Rtm_Field;

pragma page;
procedure Put_Rtm_Field (Field: in Rtm_Form.Fields;
Field_Value: in String := ") is

--/ Description:
--/ Modifies the value of a field on the RTM interface form.
--/ It then presents the information to the user. Since
--/ the rtm_command is input only, it is implemented as a nop.
--/ Likewise for the form option.
--/
--/ Parameter Description:
--/ field -> The name of the field on the RTM interface form
--/ in which to store the data.
--/ field_value -> The data to be stored in the field.
--/
--/ Notes:
--/ none
begin
  case Field is
    when Message_Field_A =>
      FormExecutor.ModifyField (Form => Rtm_Message_Form,
      Field => "message-a",
      Value => Field_Value);
      FormExecutor.PresentForm (Form => Rtm_Message_Form);
    when Message_Field_B =>
      FormExecutor.ModifyField (Form => Rtm_Message_Form,
      Field => "message-b",
      Value => Field_Value);
      FormExecutor.PresentForm (Form => Rtm_Message_Form);
    when Rtm_Command =>
      null;
    when Form =>
      null;
  end case;
exception
  when others =>
    null;
end Put_Rtm_Field;

pragma page;
procedure Clear_Rtm_Field (Field: in Rtm_Form_Fields := Form) is

-- Description:
Blanks out the current value of a field on the RTM interface form.

-- Parameter Description:
field -> The name of the field on the RTM interface form to blank.

-- Notes:
none

begin
  Blank_Line: String(1..80) := (1..80 => '');
  begin
    case Field is
      when Message_Field_A =>
        Form_Executor.Modify_Field (Form => Rtm_Message_Form,
                                   Field => "message-a",
                                   Value => Blank_Line);
      when Message_Field_B =>
        Form_Executor.Modify_Field (Form => Rtm_Message_Form,
                                   Field => "message-b",
                                   Value => Blank_Line);
      when Rtm_Command =>
        Form_Executor.Modify_Field (Form => Rtm_Prompt_Form,
                                   Field => "rtm_command",
                                   Value => Blank_Line);
      when Form =>
        Clear_Rtm_Field (Message_Field_A);
        Clear_Rtm_Field (Message_Field_B);
        Clear_Rtm_Field (Rtm_Command);
      end case;
    exception
      when others => null;
    end Clear_Rtm_Field;

pragma page;
procedure Set_Input_Mode (Next_Mode: In Input_Mode := Command_Mode) Is

/** ---------------------------------------------------------------------**/
/** Description: **
/** Select the next input mode for the RTM interface form. This **
/** interface allows the user to modify the package parameter which **
/** toggles the interface between synchronous and asynchronous I/O. **
/** **
/** Parameter Description: **
/** Next_mode -> Needed interface mode to the user. **
/** **
/** Notes: **
/** none **
/** **---------------------------------------------------------------------**/

begin
    Current_Input_Mode := Next_Mode;
end Set_Input_Mode;

end Rtm_Form;
pragma page;
Module Name:
Define RTM_CLI

Module Type:
Package Specification

Module Purpose:
This package provides the interface to the command line interpreter of the RTM.

Module Description:
The interface to the command line interpreter (CLI) is composed of three parts:
1. An enumeration type which names all the current commands available.
2. An instantiation of a generic parser package.
3. A general-purpose subroutine to get arguments from the user's command line.

Basically, the CLI is set up to operate as follows:
1. RTM main procedure:
   a. reads the user's command line
   b. parses the line for syntactic correctness using the rtm_cli defined below
2. Each routine that implements a command is responsible for:
   a. fetching all the arguments for the command
   b. checking the arguments for semantic correctness.

The legal RTM commands and their arguments (defined in the body) are:
Check (page => <page name>);
Edit ();
Quit ();
Read (name => <name>);
Set (name => <name>, value => <value>);
Start (page => <page name>, update_rate => <rate in secs>);
Stop (page => <page name>);

Details about how the commands are used and what they do are discussed in the RTM User's Manual.

References:
Design Documents:
Real-Time Monitor Requirements
Real-Time Monitor Design

User's Manual:
RTM User's Manual

Testing and Validation:
none

Notes:

The command `update_active_pages` is used internally by the RTM and does not (and never should) have a definition in the body such that the user can invoke the command.

Modification History:

16Apr87  rlv  created

Distribution and Copyright Notice:

TBD

Disclaimer:

"This work was sponsored by the Department of Defense. The views and conclusions contained in this document are solely those of the author(s) and should not be interpreted as representing official policies, either expressed or implied, of Carnegie Mellon University, the U.S. Air Force, the Department of Defense, or the U.S. Government."

pragma page;
with Standard_Interface;
-- Use the generic package "command_line" to instantiate the parser for
-- the RTM command language
--
package Define_Rtm_Cli is
-- Define the commands recognized by the RTM.
--
type Rtm_Command_Representation is
  (Check, Edit, Quit, Read, Set, Start, Stop, Update_Active_Pages);
-- Create the CLI to parse the commands defined above.
--
package Rtm_Cli is new Standard_Interface.Command_Type
  (Rtm_Command_Representation);
-- Create the structure to hold the argument definitions for
-- all the commands. Use the "define_argument" entry.
--
  Rtm_Commands: Rtm_Cli.Process_Handle_Array;
pragma page;
procedure Get_Argument (Command: In Rtm_Command_Representation;
   Argument_Name: In String;
   Argument_Value: In out String);

-- Description:
-- Retrieve an argument from a command line entered by the
-- user. If the user defaults an argument, then the default
-- value is returned.

-- Parameter Description:
-- command -> The command which the user entered and is currently
-- being processed.
-- argument_name -> The name of the argument that is needed
-- in the command line.
-- argument_value -> The value entered by the user or the default
-- value for the argument.

procedure Clear_Command_Line (Command: In Rtm_Command_Representation);

-- Description:
-- Used to reset the parser after a command had been parsed and
-- processed.

-- Parameter Description:
-- Command -> The name of the command which was just parsed and
-- needs to be reset.

end Define_Rtm_CLI;
pragma page;
with Standard_Interface;
-- Use the "set_tool" and "define_process" services.

package Define_Rtm_Cli is
-- Instantiate the package needed to define the arguments to the commands.
-- Use the Define_Argument entry in package String_Arguments.
--
package Sa is new Standard_Interface.String_Argument("string");

package Si renames Standard_Interface;
--
-- Visible Procedures
--
procedure Get_Argument (Command: in Rtm_Command_Representation;
    Argument_Name: in String;
    Argument_Value: in out String) is separate;

procedure Clear_Command_Line (Command: in Rtm_Command_Representation)
    is separate;

pragma page;
Definertmcli package body

Description:
What follows is the definition of all the commands and their arguments. This is done in two steps:
1. Define a process for the command.
2. Define each of the arguments for the command.
This process is repeated for every user command in the RTM.
The commands are defined in alphabetical order for ease of maintenance.

begin
-- Identify the tool being created to the parser package, this isn't used anywhere, but is required by the parser package
--
Si.SetTool_Identifier("RTM");
-- Define the check command --

--
Si.DefineProcess (Name => Rtm_Command_Representation'Image(Check),
                 Proc => Rtm_Commands(Check));

Sa.DefineArgument(Proc => Rtm_Commands(Check),
                  Name => "page",
                  Default => "",
                  Help => "See User's Manual");
-- Define the edit command --

--
Si.DefineProcess (Name => Rtm_Command_Representation'Image(Edit),
                 Proc => Rtm_Commands(Edit));

-- Define the quit command --

--
Si.DefineProcess (Name => Rtm_Command_Representation'Image(Quit),
                 Proc => Rtm_Commands(Quit));

-- Define the read command --

--
Si.DefineProcess (Name => Rtm_Command_Representation'Image(Read),
                 Proc => Rtm_Commands(Read));

Sa.DefineArgument(Proc => Rtm_Commands(Read),
                  Name => "name",
                  Help => "See RTM User's Manual");
-- Define the set command --

Si.Define_Process (Name => Rtm_Command_Representation'Image(Set),
     Proc => Rtm_Commands(Set));

Sa.Define_Argument(Proc => Rtm_Commands(Set),
     Name => "name",
     Help => "See RTM User's Manual");

Sa.Define_Argument(Proc => Rtm_Commands(Set),
     Name => "values",
     Help => "See RTM User's Manual");

-- Define the start command --

Si.Define_Process (Name => Rtm_Command_Representation'Image(Start),
     Proc => Rtm_Commands(Start));

Sa.Define_Argument(Proc => Rtm_Commands(Start),
     Name => "page",
     Default => "", 
     Help => "See RTM User's Manual");

Sa.Define_Argument(Proc => Rtm_Commands(Start),
     Name => "update_rate",
     Default => "2.0", 
     Help => "See RTM User's Manual");

-- Define the stop command --

Si.Define_Process (Name => Rtm_Command_Representation'Image(Stop),
     Proc => Rtm_Commands(Stop));

Sa.Define_Argument(Proc => Rtm_Commands(Stop),
     Name => "page",
     Default => "", 
     Help => "See RTM User's Manual");

end Define_Rtm_Cli;
pragma page;
with String_Pkg;

-- Use "string_type" to create a dynamic length for interfacing to
-- the RTM_Cli.
-- Use "length" and "value" to convert the dynamic strings back into
-- normal Ada strings; "length" returns the number of characters in
-- a dynamic string, and "value" returns the characters in the string
-- as a simple Ada string.

separate (Define_Rtm_Cli)
procedure Get_Argument (Command: in Rtm_Command_Representation;
                     Argument_Name: in String;
                     Argument_Value: in out String) is
............................................................

Description:
--- Retrieve an argument from a command line entered by the
--- user. If the user defaults an argument, then the default
--- value is returned. The main reason for the existence of this
--- procedure is twofold:
--- 1. To convert the dynamic strings used in the parser to
---     the regular strings used by Ada.
--- 2. To blank out any stray characters which may be lingering
---     in the argument_value string.

Parameter Description:
--- command -> The command that the user entered and is currently
---     being processed.
--- argument_name -> The name of the argument that is needed
---     in the command line.
--- argument_value -> The value entered by the user or the default
---     value for the argument.

Notes:
--- none
............................................................

package Sp renames String_Pkg;
   Interface_String: Sp.String_Type;

   Blanks: String(1..256) := (1..256 => ' ');
begin
   Argument_Value := Blanks(Argument_Value'range);
   Interface_String := Sa.Get_Argument(Proc => Rtm_Commands(Command),
                                       Name => Argument_Name);
   Argument_Value (1..Sp.Length(Interface_String)) :=
       Sp.Value(Interface_String);
exception
   when others =>
      RAISE ;
end Get_Argument;
pragma page;
procedure Clear_Command_Line (Command: In Rtm_Command_Representation) is

--/ Description:
--/ Used to reset the parser after a command had been parsed and
--/ processed.

--/ Parameter Description:
--/ Command -> The name of the command that was just parsed and
--/ needs to be reset.

begin
   S1.Redefine_Process (Proc => Rtm_Commands(Command));
end Clear_Command_Line;
pragma page;
package Page_Processor is

pragma page;
procedure Check_Page;

--/ Description:
--/ This command checks a page for consistency, i.e., it checks
--/ a page's variables against the variable database to insure
--/ that each one is accessible to the RTM.
--/
--/ Parameter Description:
--/ none
--/ ........................................................................

pragma page;
procedure Start_Page;

/* Description:
* Allows the user to select a page for display or history
* collection at a periodic update rate.
*/

/* Parameter Description:
* none
*/

pragma page:
procedure Stop_Page;

--| Description:
--| Allows the user to terminate use of a page.
--
--| Parameter Description:
--| none
--

pragma page;
procedure Update_Pages;

--/ Description:
--/ This entry is used by the monitor to update any pages that
--/ are currently active. It is not a user command.
--/
--/ Parameter Description:
--/ none
--/........................................................................

end Page_Processor;
pragma page;
Module Name: Page_Processor

Module Type: Package Body

Module Description:
This package contains the interface to all the user commands (dealing with pages) in the RTM User's Manual. Due to the nature of the parser used in the RTM, none of the command procedures need arguments because parser software internally maintains all arguments for the last parsed line. This package also defines structures needed to process the pages (defined below).

References:
Design Documents:
- Real-Time Monitor Requirements
- Real-Time Monitor Design

User's Manual:
- RTM User's Manual

Testing and Validation:
none

Notes:
none

Modification History:
16Apr87 rvs created

Distribution and Copyright Notice:
TBD

Disclaimer:
"This work was sponsored by the Department of Defense. The views and conclusions contained in this document are solely those of the author(s) and should not be interpreted as representing official policies, either expressed or implied, of Carnegie Mellon University, the U.S. Air Force, the Department of Defense, or the U.S. Government."

pragma page;
with Calendar, use Calendar.
-- Use type "time"

with Dialogue_Manager,
-- Use type "variable_identifier"

with Form_Executor,
-- Use type "form_ptr"

with Form_Manager
-- Use types "field_mode", "field_access" and "field_length"

with Lists
-- Use generic package "lists"
-- Use type "list"
-- Use service "create"

with Text_io,
-- Use generic package "fixed_io"

with Define_Rtm_Cli, use Define_Rtm_Cli;

package Page_Processor is
-- Instantiate fixed_io for use in converting strings to type duration

package Duration_io is new Text_io.Fixed_io(Duration);
-- A page is composed of page information and variables (aka form
-- field names). Processing a page once it has been
-- activated for display requires that the RTM know the name of
-- each variable on the page. The information needed about each
-- variable on a page is:
-- variable_name -> The name of the variable, which corresponds
-- to the name of a field on the form representation
-- of the page where the data are to be displayed.
-- vid -> A pointer into the variable database, where all the known
-- data on the variable are kept.
-- display_length -> The size of the output field (on the form) for
-- this variable.

subtype Variable_Name_Representation is String(1..80);
type Page_Field_Representation is record
  Variable_Name: Variable_Name_Representation;
  Vid: Dialogue_Manager.Variable_Identifier;
  Display_Length: Integer;
end record;

-- A page is composed of miscellaneous page data:
-- the_page -> A pointer to the form representation of
-- the page. This is used by the form_executor
-- system when updating and displaying the page.
-- page_name -> The user's name for the page.
next_update_time -> The time when the page is to be refreshed on the output device.
refresh_rate -> The rate at which the page is to be refreshed
After the page data comes the data about each of the variables on the page (documented above). Since the number of variables on a page is variable, a linked list is used to tie them all together for each page:
page_fields -> The linked list of all the legal variables on the page.

package Field_Lists is new Lists (Page_Field_Representation);
subtype Page_Name_Representation is String(1..80);
type Page_Representation is record
  The_Page: Form_Executor.Form_Ptr;
  Page_Name: Page_Name_Representation;
  Next_Update_Time: Calendar.Time;
  Refresh_Rate: Duration;
end record;

For each active page, there is one page_representation record. All the active pages are kept in an array and added and deleted according to page counts shown below.

Maximum_Number_Of_Active_Pages: constant Integer := 2;
Current_Number_Of_Active_Pages: Integer := 0;
Active_Page: array (1..Maximum_Number_Of_Active_Pages) of Page_Representation;

pragma page;
-- Internal procedures

procedure Get_Fields (Field_Pointer: in Form_Manager.Field_Access;
                     The_Variable_Name: in out Variable_Name_Representation;
                     The_Mode: in out Form_Manager.Field_Mode;
                     The_Length: in out Form_Manager.Field_Length) is separate;

function Setup_Page (The_Page: in String;
                     Collection_Rate In Duration)
                      return Integer is separate;

-- Visible procedures

procedure Check_Page is separate;
procedure Update_Pages is separate;
procedure Start_Page is separate;
procedure Stop_Page is separate;

end Page_Processor;
pragma page;
with Variable_Database;
   -- Use the type "the_variable".
   -- Use the service "find".
   ...
with Rtm_Form; use Rtm_Form;
   -- Use the service "put_rtm_field".
   ...
with Form_Manager; use Form_Manager;
   -- Use the types "form_access", "field_access", "field_length"
   -- and "field_mode".
   -- Use the exception "field_not_found".
   -- Use the services "Get_first_field", "Get_next_field" and "modify_field".
   ...
with Form_Executor;
   -- Use the services "access_form" and "modify_field".
   -- Use the exception "form_access_error".
   ...
with Dialogue_Manager;
   -- Use the type "variable_identifier".
   -- Use the exception "variable_not_found".
   -- Use the service "get_identifier".

separate (Page_Processor)
procedure Check_Page is
   -- Description:
   -- This command checks a page for consistency, i.e., it checks
   -- a page's variables against the variable database to insure
   -- that each one is accessible to the RTM.
   --
   -- The functioning of this module is very similar to Setup_page.
   -- The main difference is that this module doesn't build the
   -- actual list of variables; it simply checks the field_mode
   -- and existence of each variable, giving the user error messages
   -- where appropriate.
   --
   -- Parameter Description:
   -- none
   --
   -- Notes:
   -- 1. All the arguments are obtained from the parser when needed.
   --
   -- 2. The exception form_manager.field_not_found is raised
   -- by the form_manager when the end of the form is reached.
   -- This kicks the module out of the loop which builds the variable
   -- list for the page and is the normal exit point for the
   -- module.
   --
   -- 3. The exception form_executor.form_access_error is raised
   -- when the form_manager cannot access the user-requested
   -- page. It is propagated out to the caller to indicate
   -- a bad page.
   --
package Cli renames Define_Rtm_Cli;

The_Variable_Name: Variable_Name_Representation;
The_Working_Page: Page_Name_Representation;
A_Variable: Variable_Database.The_Variable;
The_Page: Form_Manager.Form_Access;
Error_Count: Integer := 0;

-- declarations needed to access the data in the form_manager
Field_Pointer: Form_Manager.Field_Access;
The_Length: Form_Manager.Field_Length := 1;
The_Mode: Form_Manager.Field_Mode := Constant_Text;

begin
  Cli.Get_Argument (Command => Cli.Check,
                    Argument_Name => "page",
                    Argument_Value => The_Working_Page);

  The_Page := Form_Executor.Access_Form (Pathname => The_Working_Page);

  -- Now we loop through all the fields defined for the page, obtaining
  -- the variable name and format length from the form_manager, and
  -- the variable identifier from the variable_database,
  -- as if we were building the page definition.

  Field_Pointer := Form_Manager.Get_First_Field(The_Page);
  loop
    begin
      Get_Fields (Field_Pointer, The_Variable_Name,
                 The_Mode, The_Length);
      case The_Mode is
        when Constant_Text =>
          null;
        when Input_Output =>
          Put_Rtm_Field (Message_Field_A,
                         "illegal mode for variable:" &
                         The_Variable_Name);
          Error_Count := Error_Count + 1;
        when Output_Only =>
          A_Variable := Variable_Database.Find
                        (Name => The_Variable_Name);
      end case;
    exception
      when Variable_Database.Variable_Not_Found =>
        Put_Rtm_Field (Message_Field_A,
                       "Variable not found:" & The_Variable_Name);
        Error_Count := Error_Count + 1;
      when others =>
        Put_Rtm_Field (Message_Field_B,"exception raised in check...");
    end;
    Field_Pointer := Form_Manager.Get_Next_Field(Field_Pointer);
  end loop;
exception
  when Form_Manager.Field_Not_Found =>
    Put_Rtm_Field (Message_Field_B,"Check completed with " &
Integer Image(Error_Count) & " errors";

when Form_Executor.Form_Access_Error =>
  Put_Rtm_Field (Message_Field_A, "Error in accessing page: " &
  The_Working_Page);

end Check_Page;
pragma page;
with Rtm_Form, use Rtm_Form;
-- Use the service "put_rtm_field".
--
separate (Page_Processor)
procedure Start_Page is

-- Description:
-- Allows the user to select a page for display or history
-- collection at a periodic update rate. The processing is:
-- 1. Get the command arguments.
-- 2. Set up the page definition.
-- 3. Set up the input mode.

-- Parameter Description:
-- none

-- Notes:
-- 1. All command arguments are obtained from the parser as needed
-- 2. The exceptions form_executor.invalid_form and
-- form_executor.form_access_error are raised by Setup_page.
-- and indicate that a bad page was specified by the user

package Cli renames Define_Rtm_Cli;

End_Of_Value: Integer;
Collection_Rate: Duration;
Field_Position: Field_List.Listiter;
The_Page_Fields: Page_Field_Representation;
Update_Rate: String (1..80);
Page_Name: Page_Name_Representation;
Current_Time: Calendar.Time := Calendar.Clock;
Page_Number: Integer;

begin
  case Current_Number_Of_Active_Pages is
    
    when Maximum_Number_Of_Active_Pages =>
      Put_Rtm_Field (Message_Field_A,
                    "Maximum number of active pages already in use"),
      Put_Rtm_Field (Message_Field_B,
                    "a Stop command must be issued first");

    when others =>

    Get the Start command arguments entered by the user and build internal
    page definition.
    Cli.Get_Argument (Command => Cli.Start,
Argument_Name => "page",
Argument_Value => Page_Name);
Cli.Get_Argument (Command => Cli.Start,
Argument_Name => "update_rate",
Argument_Value => Update_Rate);
Duration.Get (From => Update_Rate,
    Item => Collection_Rate,
    Last => End_Of_Value);

--d
collection_rate := duration(integer value(update_rate));
--d
Page_Number = Setup_Page (Page_Name.Collection_Rate);
Active_Page(Page_Number).Refresh_Rate := Collection_Rate.
Active_Page(Page_Number).Next_Update_Time :=
    Current_Time;

-- Since we're starting a page, put the user input mode into
-- SCREEN_MODE, which allows the screen to be updated asynchronously
-- from user input:
--
    Rtm_Form.Set__Input_Mode (Rtm_Form.Screen_Mode);
end case;
exception
when Text_io.Data_Error=>
    Put_Rtm_Field (Message_Field_A,
        "Bad update rate, reenter in xxx format");
when Form_Executor.Invalid_Form
    Form_Executor.Form_Access_Error =>
        null;
end Start_Page;
pragma page;
with CaseInsensitiveStringComparison;
-- Use the service "equal";

with Terminal_Interface;
-- Use the service "clear_screen";

with Rtm_Form; use Rtm_Form;
-- Use the service "put_rtm_field";

separate (Page_Processor)
procedure Stop_Page is
... ...

--/ Description:
-- Allows the user to terminate use of a page. The processing
-- is:
-- 1. Search the array of active pages for the user-specified
--    page.
-- 2. If it's not found, then issue an error message.
-- 3. Otherwise,
--    a. loop thru all the variables on the page
--    b. deactivate each one (i.e., remove them from
--        the list of variables on which data is collected).
--    c. destroy the page definition.
--
--/ Parameter Description:
-- none
--
--/ Notes:
-- 1. All command arguments are obtained from the parser as needed.
--
-- 2. Page_mismatch error renames constraint error; this is used
--    when searching the active page array for the page to stop.
--    The constraint_error is raised when the loop exceeds the
--    dimension of the array, which says that no active page
--    matches the page which the user wishes to stop, i.e.,
--    a "page_mismatch".
--
......

package Cisc renames CaseInsensitiveStringComparison;

package Cli renames Define_Rtm_Cli;

Field_Position: Field_Lists.ListIter;
Page_To_Stop: Page_Name_Representation;
The_Page_Fields: Page_Field_Representation;
Page_Number: Integer := 1;
Usage: Dialogue_Manager.io_Usage := Dialogue_Manager.Read;

Page_Mismatch: exception renames Constraint_Error;

begin
  case Current_Number_Of_Active_Pages Is
  ..
  when there are no pages active, then there's nothing to do but
  inform the user.
when 0 =>
    Put_Rtm_Field (Message_Field_A,"no active pages to stop");
when others =>

-- Get the name of the page to stop, and compare it against the current
-- active pages; if there's no match, issue an error message (a constraint
-- error is raised when we reach the end of the array), which we have
-- conveniently renamed as a page_mismatch exception.

    Cli.Get_Argument (Command => Cli.Stop,
        Argument_Name => "page",
        Argument_Value => Page_To_Stop);
    while not Cisc.Equal(Active_Page(Page_Number).Page_Name, Page_To_Stop) loop
        Page_Number := Page_Number + 1;
    end loop;

-- If everything is correct, then we iterate through the active variables
-- for the current page, deactivating them in the process.

    Field_Position := Field_Lists.MakeListIter(
        Active_Page(Page_Number).Page_Fields);
    while Field_Lists.More(Field_Position) loop
        Field_Lists.Next(Field_Position, The_Page_Fields);
        Dialogue_Manager.Deactivate (The_Page_Fields.Vid, Usage);
    end loop;

-- When all the variables are deactivated, we destroy the active page data,
-- since we just deactivated.

    Active_Page(Page_Number).Page_Name (1..10) := " ";
    Field_Lists.Destroy(Active_Page(Page_Number).Page_Fields);
    Form_Executor.Release_Form
        (Form => Active_Page(Page_Number) The_Page);
    Terminal_Interface.Clear_Screen;

-- Move the reset of active pages into the space just vacated by the
-- deletion from the active page list

    for New_Page_Number In Page_Number Current_Number..0' Active_Pages' loop
        Active_Page(New_Page_Number) := Active_Page(New_Page_Number - 1);
    end loop;
    Current_Number_Of_Active_Pages := Current_Number_Of_Active_Pages - 1;
    if Current_Number_Of_Active_Pages = 0 then
        Rtm_Form.Set_Input_Mode (Rtm_Form Command_Mode 0);
    end if;
end case;

exception

-- Handle the error that occurs when the user tries to stop a page which
-- isn't active at the moment.
when Page_Mismatch =>
    Put_Rtm_Field (Message_Field_A,
        "Page not currently active: " & Page_To_Stop);
when others =>
    null;
end Stop_Page;
pragma page;
package Cli renames Define_Rtm_Cli;

Field_Position: Field_Lists.ListIter;
The_Page_Fields: Page_Field_Representation;
The_Value: Dialogue_Manager.ValueString;
Update_Rate: String (1..80);
Page_Number: Integer := 1;
Current_Time: Calendar.Time;

begin
  case Current_Number_Of_Active_Pages Is
      when 0 =>
        null;
      when others =>
        while Page_Number <= Current_Number_Of_Active_Pages loop

        -- Determine if it's time to update this page on the display.

        Current_Time := Calendar.Clock;
        if Current_Time >= Active_Page(Page_Number).Next_Update_Time then

          -- If the time is right, then we create a list iterator,

          Field_Position := Field_Lists.MakeListIter(
            Active_Page(Page_Number).Page.Fields);

          -- and iterate over all the variables on the current active page,
          -- extracting the value from the dialogue_manager and updating
          -- the values in the page's internal form representation.

          null
        end if;
      end while
  end case;
end;
while Field_Lists.More(Field_Position) loop
  Field_Lists.Next(Field_Position, The_Page_Fields);
  The_Value := Dialogue_Manager.GetValue ( 
    The_Page_Fields.Id, 
    The_Page_Fields.Display_Length);
  FormExecutor.Modify_Field ( 
    Form => Active_Page(Page_Number).The_Page, 
    Field => The_Page_Fields.Variable_Name, 
    Value => The_Value(1..The_Page_Fields.Display_Length));
end loop;
FormExecutor.Present_Form (Active_Page(Page_Number).The_Page, 
  Active_Page(Page_Number).Next_Update_Time := Current_Time + 
  Active_Page(Page_Number).Refresh_Rate);
end if;
  Page_Number := Page_Number + 1;
end loop;
end case;
exception
  when others =>
    null;
end Update_Pages;
pragma page;
with Form_Manager, use Form_Manager;

-- Use the types "field_name", "field_mode", "field_length",
-- "field_name", "field_position", "field_renditions", "char_type",
-- "field_value".
-- Use the service "get_field_info".

separate (Page_Processor)
procedure Get_Fields (Field_Pointer: in Form_Manager.Field_Access,
        The_Variable_Name: in out Variable_Name_Representation,
        The_Mode: in out Form_Manager.Field_Mode,
        The_Length: in out Form_Manager.Field_Length) is

-- Description:
-- This routine allows the caller to get only the relevant data
-- from the form_manager's definition of a page, since
-- not all the data which the form_manager provides are used.
-- It also does conversion/cleanup work on the data.
--
-- The basic operation of this module is:
-- 1. Use the forms management subsystem to get all the data
--     about the desired field.
-- 2. Blank out any garbage in the variable name.
-- 3. Return the data to the caller.
--
-- Parameter Description:
-- field_pointer -> The pointer to field about which the
--                 information is needed. It is the job
--                 of the caller to determine which field
--                 is of interest.
-- the_variable_name -> The variable name found in the field
--                     (in the form_manager, it is the name of the
--                     field; to the RTM, it is the variable name)
-- the_mode -> The I/O mode of the variable:
--            INPUT_OUTPUT or OUTPUT_ONLY
-- the_length -> The number of characters allowed on the form
--              for displaying the data of the variable.
--
-- Notes:
-- none

Blanks: Variable_Name_Representation := (others => ' ');

-- Declarations needed to access the data in the form_manager:
The_Field_Name: Form_Manager.Field_Name;
The_Location: Form_Manager.Field_Position;
The_Rendition: Form_Manager.Field_Rendtions;
The_Limits: Form_Manager.Char_Type;
The_Initial_Value: Form_Manager.Field_Value;
The_Current_Value: Form_Manager.Field_Value;

begin
-- Get all the information for the next field on the page from the
-- form manager.

Form_Manager.Get_Field_Info (Field => Field_Pointer,
Name => The_Field_Name,
Position => The_Location,
Length => The_Length,
Rendition => The_Rendition,
Char_Limits => The_Limits,
Init_Value => The_Initial_Value,
Value => The_Current_Value,
Mode => The_Mode),

-- Clean up the variable name so that there's no garbage variable name we return to the caller.

The_Variable_Name := Blanks;
The_Variable_Name(T..The_Field_Name'Last) := The_Field_Name;

end Get_Fields;
pragma page;
with Variable_Database,
   -- Use the type "the_variable".
   -- Use the exception "variable_not_found".
   -- Use the service "find".

with Rtm_Form; use Rtm_Form,
   -- Use the service "put_rtm_field".

with Form_Manager; use Form_Manager
   -- Use the types "form_access", "field_access", "field_length"
   -- and "field_mode"
   -- Use the exception "field_not_found"
   -- Use the services "Get_first_field", "Get_next_field" and "modify_field".

with Form_Executor;
   -- Use the services "access_form" and "modify_field".
   -- Use the exception "form_access_error".

with Dialogue_Manager;
   -- Use the type "variable_identifier" and "ic_usage".
   -- Use the exception "variable_not_found".
   -- Use the service "activate".

separate (Page_Processor)
function Setup_Page (The_Page: in String;
   Collection_Rate: in Duration) return Integer is
   ....................................................

   Description:
   This module takes a page name and uses services from the
   form_manager to build the internal representation of a page.

   The processing performed by this module is:
   1. Check to see that the page exists, and if not,
      issue an error to the user and exception out.
   2. When the page exists, the module loops through
      all the fields on the form, building the list
      of page variables from the INPUT_OUTPUT and
      OUTPUT_ONLY fields and activating the variables
      for data collection.

   Parameter Description:
   - the_page  -> The user's name for the page to be invoked.
   - return   -> The index into the active page array where the
                 page definition has been built.

   Notes:
   1. The exception form_manager.field_not_found is raised
      by the form_manager when the end of the form is reached.
      This kicks the module out of loop which builds the variable
      list for the page, and is the normal exit point for the
      module.
   2. The exception form_executor.form_access_error is raised
      when the forms management subsystem cannot access the
      user-requested page. It is propagated out to the caller
      to indicate a bad page.
package Cli renames Define_Rtm_Cli;

Current_Time: Calendar.Time := Calendar.Clock;
Page_Number: Integer;
A_Variable: Variable_Database.The_Variable;
Active_Variable: Dialogue_Manager.Variable_Identifier;
Temporary_Form: Form_Manager.Form_Access;
The_Variable_Name: Variable_Name_Representation;

-- Declarations needed to access the data in the form_manager:
Field_Pointer: Form_Manager.Field_Access;
The_Length: Form_Manager.Field_Length := 1;
The_Mode: Form_Manager.Field_Mode := Constant_Text;
begin

-- Set up an active page definition for the page selected by the user.
-- 1. Load the form definition for the page selected by the user.
-- 2. Increment the count of the number of active pages.
-- 3. Store the pointer to the form definition and the name of the page.
Temporary_Form := FormExecutor.Access_Form (Pathname => The_Page);
Current_Number_Of_Active_Pages := Current_Number_Of_Active_Pages + 1;
Page_Number := Current_Number_Of_Active_Pages;
Active_Page(Page_Number).The_Page := Temporary_Form;
Active_Page(Page_Number).Page_Name := The_Page;

-- Once all the form level items are set up,
-- we loop through all the fields defined for the form/page, obtaining
-- the variable name and format length from the form_manager, and
-- the variable_identifier from the variable_database; using this, we
-- activate the variable for data collection and build the page field
-- definition by creating a list of
-- (variable_name, variable_identifier, format length) records.

Field_Pointer := Form_Manager.Get_First_Field
(Active_Page(Page_Number).The_Page);
loop
    begin
        Get_Fields (Field_Pointer, The_Variable_Name,
The_Mode, The_Length);
case The_Mode is
    ..
    ..
Constant_text fields are trim items on the form and
not of interest to the RTM.
when Constant_Text =>
    null ;
.. Input_output fields contain variables of interest, but were
entered inappropriately by the user, so we change the mode and

use the fields appropriately.

when Input_Output =>
   Form_Manager.Modify_Field_Mode (Field => Field_Pointer,
      Mode => Output_Only);
   AVaniable := Variable_Database.Find
      (Name => The_Variable_Name);
   Active_Variable := Dialogue_Manager.Activate
      (The_Variable_Name,
         Collection_Rate,
         Current_Time,
         Dialogue_Manager.Read);
   FieldLists.Attach (Active_Page(Page_Number).Page_Fields,
      (Variable_Name => The_Variable_Name,
         Vid => Active_Variable,
         Display_Length => The_Length));

-- These fields also contain variables of interest to the user,
-- but are a little simpler, since they are already in the
-- proper mode.

when Output_Only =>
   AVaniable := Variable_Database.Find
      (Name => The_Variable_Name);
   Active_Variable := Dialogue_Manager.Activate
      (The_Variable_Name,
         Collection_Rate,
         Current_Time,
         Dialogue_Manager.Read);
   FieldLists.Attach (Active_Page(Page_Number).Page_Fields,
      (Variable_Name => The_Variable_Name,
         Vid => Active_Variable,
         Display_Length => The_Length));

end case;

exception

   when Variable_Database.Variable_Not_Found | Dialogue_Manager.Variable_Not_Found =>
      Form_Executor.Modify_Field
      (Form => Active_Page(Page_Number).The_Page,
         Field => The_Variable_Name,
         Value => "error................................");
      Put_Rtm_Field (Message_Field_A, "Variable not found: " &
         The_Variable_Name);

   when others => --debug
      Put_Rtm_Field (Message_Field_B,"exception raised in setup...");

end;

Field_Pointer := Form_Manager.Get_Next_Field(Field_Pointer);

end loop;

exception

   when Form_Manager.Field_Not_Found =>
      RETURN Page_Number;

58 October 1987
when FormExecutor/Form_Access_Error =>
  Put_Rtm_Field (Message_Field_A, "Error in accessing page: " & The_Page);
  RAISE;
end Setup_Page;
pragma page;
Module Name:
Parameter_Manager

Module Type:
Package Specification

Module Purpose:
Manages the reading and writing of single Ada variables
independently of displaying pages.

Module Description:
This package manages the reading and writing of single Ada
variables without interfering with active display pages.

References:
Design Documents:
Real-Time Monitor Requirements
Real-Time Monitor Design

User's Manual:
RTM User's Manual

Testing and Validation:
none

Notes:
none

Modification History:
08Apr87 rlvs Created

Distribution and Copyright Notice:
TBD

Disclaimer:
"This work was sponsored by the Department of Defense.
The views and conclusions contained in this document are
solely those of the author(s) and should not be interpreted as
representing official policies, either expressed or implied,
of Carnegie Mellon University, the U.S. Air Force,
of the Department of Defense, or the U.S. Government."

pragma page;
package Parameter_Manager is

  procedure Read;
  --| Description:
  --| Extracts the value of a variable from application memory.
  --
  --| Parameter Description:
  --| none
  --

pragma page;


procedure Set;

--| Description:
--| Deposits the user-supplied value into application memory.
--
--| Parameter Description:
--| none
--
end Parameter_Manager;
pragma page:
Module Name:
Parameter_manager;

Module Type:
Package Body

Module Description:
This package implements two commands:

Read (name => <variable>);
Set (name => <variable>, value => < >);

The package does very little except group the single variable
operations together.

References:
Design Documents:
Real-Time Monitor Requirements
Real-Time Monitor Design

User's Manual:
RTM User's Manual

Testing and Validation:
none

Notes:
none

Modification History:
30Apr87 rvs created

Distribution and Copyright Notice:
TBD

Disclaimer:
"This work was sponsored by the Department of Defense.
The views and conclusions contained in this document are
solely those of the author(s) and should not be interpreted as
representing official policies, either expressed or implied,
of Carnegie Mellon University, the U.S. Air Force,
the Department of Defense, or the U.S. Government."

pragma page;
package Parameter_Manager is

-- Internal procedures

function Left_Justify (Display_Value In String;
return String is separate;

-- Visible procedures

procedure Read is separate
procedure Set is separate;

end Parameter_Manager;

pragma page,
with Dialogue_Manager;
  - Use the types "variable_identifier" and "value_string".
  - Use the services "activate", "get_value", "get_identifier" and "deactivate".

with Define_Rtm_Cli;
  - Use the service "get_argument".

with Rtm_Form, use Rtm_Form;
  - Use the service "put_rtm_field".

with Calendar, use Calendar;
  - Use the type "time".
  - Use the service "clock".

separate (Parameter_Manager)
procedure Read is
  /* Description:
   - Extracts the value of a variable from application memory.
   - The processing involved is straightforward:
   1. Get the variable to be read from the user's command line.
   2. If the variable is not in the database, then issue an error message and get out.
   3. Otherwise:
      a. activate the variable for input,
      b. read the value,
      c. display the value to the user,
      d. and deactivate the variable (since we're finished with it).
  */

  /* Parameter Description:
   - none
  */

  /* Notes:
   - none
  */

package Cli renames Define_Rtm_Cli,

  The_Variable: Dialogue_Manager Variable_identifier,
  The_Value: Dialogue_Manager Value_string,
  Variable_Name: String (1..80),
  Current_Time: Calendar Time = Clock,
begin
  -- Get the variable name and verify that it's legal.
  "Cli Get_Argument (Command => Cli_Read,
   Argument_Name => "name",
   Argument_Value => Variable_Name),

  -- Activate the variable, extract the data, and display it to the user
The Variable := Dialogue_Manager.Activate
 (Name => Variable_Name,
  Usage => Dialogue_Manager.Read,
  Starting_Time => Current_Time,
  Rate => 0.0);

The_Value := Left_Justify(Discussion_Manager.Get_Value(Vid=>The_Variable));

Put_Rtm_Field(Field => Message_Field_A,
   Field_Value => "Variable: " & Variable_Name);

Put_Rtm_Field(Field => Message_Field_B,
   Field_Value => "has the value: " & The_Value);

Dialogue_Manager.Deactivate(Vid => The_Variable,
   Usage => Dialogue_Manager.Read);

exception
when Dialogue_Manager.Variable_Not_Found =>
   Put_Rtm_Field(Field => Message_Field_A,
      Field_Value => "Variable not Found: " & Variable_Name);

when others =>
   Put_Rtm_Field(Field => Message_Field_A,
      Field_Value => "Unhandled exception in param_man");

end Read;

pragma page;
with Dialogue_Manager;
    -- Use the types "variable_identifier" and "value_string".
    -- Use the services "activate", "get_value", "get_identifier" and
    -- "deactivate".

with Define_Rtm_Cli;
    -- Use the service "get_argument".

with Rtm_Form; use Rtm_Form;
    -- Use the service "put_rtm_field".

with Calendar; use Calendar;
    -- Use the type "time".
    -- Use the service "clock".

-- separate (Parameter_Manager)

procedure Set is
    -- unsupported
    --/ Description:
    --/ Deposits the user-supplied value into application memory.
    --/ The processing involved is straightforward:
    --/ 1. Get the variable and value to write from the user's
        --/ command line.
    --/ 2. If the variable is not in the database, then
        --/ issue an error message and get out.
    --/ 3. Otherwise,
        --/ a. activate the variable for output,
        --/ b. display the status to the user,
        --/ c. and deactivate the variable (since we're finished with it).
    --/ Parameter Description:
    --/ none
    --/ Notes:
    --/ none

package Cli renames Define_Rtm_Cli;

    The_Variable: Dialogue_Manager.Variable_Identifier;
    The_New_Value: Dialogue_Manager.Value_String;
    The_Value: Dialogue_Manager.Value_String;
    Variable_Name: String (1..80);
    Current_Time: Calendar.Time := Clock;

begin
    -- Get the variable_name and value entered by the user and
    -- verify that the variable is available for i/o.
    Cli.Get_Argument (Command => Cli.Set,
        Argument_Name => "name",
        Argument_Value => Variable_Name);
    Cli.Get_Argument (Command => Cli.Set,
        Argument_Name => "values"
Argument_Value \Rightarrow \text{The}_\text{New}_\text{Value};

-- Deposit the data into application memory and display the status.
-- The Activate procedure call must precede the Set_Value procedure call,
-- since Set_Value requires a variable_identifier to function properly.

\text{The}_\text{Variable} \Rightarrow \text{Dialogue}_\text{Manager}.\text{Activate}
\hspace{1em} (\text{Name} \Rightarrow \text{Variable}_\text{Name},
\hspace{1em} \text{Usage} \Rightarrow \text{Dialogue}_\text{Manager}.\text{Write},
\hspace{1em} \text{Starting}_\text{Time} \Rightarrow \text{Current}_\text{Time},
\hspace{1em} \text{Rate} \Rightarrow 0.0);
\hspace{1em} \text{Dialogue}_\text{Manager}.\text{Set}_\text{Value} (\text{Vid} \Rightarrow \text{The}_\text{Variable},
\hspace{1em} \text{Value} \Rightarrow \text{The}_\text{New}_\text{Value});
\hspace{1em} \text{The}_\text{Value} \Rightarrow \text{Left}_\text{Justify} (\text{Dialogue}_\text{Manager}.\text{Get}_\text{Value} (\text{Vid} \Rightarrow \text{The}_\text{Variable}));
\hspace{1em} \text{Put}_\text{Rtm}_\text{Field} (\text{Field} \Rightarrow \text{Message}_\text{Field}_\text{A},
\hspace{1em} \text{Field}_\text{Value} \Rightarrow "\text{Variable} : " \& \text{Variable}_\text{Name});
\hspace{1em} \text{Put}_\text{Rtm}_\text{Field} (\text{Field} \Rightarrow \text{Message}_\text{Field}_\text{B},
\hspace{1em} \text{Field}_\text{Value} \Rightarrow "\text{now has the value: } " \& \text{The}_\text{Value});
\hspace{1em} \text{Dialogue}_\text{Manager}.\text{Deactivate} (\text{Vid} \Rightarrow \text{The}_\text{Variable},
\hspace{1em} \text{Usage} \Rightarrow \text{Dialogue}_\text{Manager}.\text{Write});
\hspace{1em} \text{exception}
\hspace{1em} \text{when} \text{Dialogue}_\text{Manager}.\text{Variable}_\text{Not}_\text{Found} \Rightarrow
\hspace{1em} \text{Put}_\text{Rtm}_\text{Field} (\text{Field} \Rightarrow \text{Message}_\text{Field}_\text{A},
\hspace{1em} \text{Field}_\text{Value} \Rightarrow "\text{Variable not Found: } " \& \text{Variable}_\text{Name});
\hspace{1em} \text{when} \text{Dialogue}_\text{Manager}.\text{Illegal}_\text{Value} \Rightarrow
\hspace{1em} \text{Put}_\text{Rtm}_\text{Field} (\text{Field} \Rightarrow \text{Message}_\text{Field}_\text{A},
\hspace{1em} \text{Field}_\text{Value} \Rightarrow "\text{Illegal value: } " \& \text{The}_\text{New}_\text{Value});
\hspace{1em} \text{Dialogue}_\text{Manager}.\text{Deactivate} (\text{Vid} \Rightarrow \text{The}_\text{Variable},
\hspace{1em} \text{Usage} \Rightarrow \text{Dialogue}_\text{Manager}.\text{Write});
\hspace{1em} \text{when} \text{others} \Rightarrow
\hspace{1em} \text{Put}_\text{Rtm}_\text{Field} (\text{Field} \Rightarrow \text{Message}_\text{Field}_\text{A},
\hspace{1em} \text{Field}_\text{Value} \Rightarrow "\text{Unhandled exception in param_man}");\text{end Set};
\hspace{1em} \text{pragma page};
separate (Parameter_Manager)

function Left_Justify (Display_Value In String) return String is

-- Description:
-- This function takes a string as input and strips off leading
-- blanks.

-- Parameter Description:
-- display_value -> String to process.
-- return -> left-justified string

-- Notes:
-- none

-- begin
  String_Length: Integer := Display_Value'Length;
  Starting_Character: Integer := String_Length;
  New_String: String(Display_Value'range) => (others => ' ');
  begin
    if Display_Value(1) /= ' ' then
      RETURN Display_Value;
    else
      while Display_Value(Starting_Character) /= ' ' loop
        Starting_Character := Starting_Character + 1;
      end loop;
      New_String(1..String_Length - Starting_Character + 1) :=
        Display_Value(Starting_Character..Display_Value'Length);
      RETURN New_String;
    end if;
  end if;
end Left_Justify,
pragma page
Module Name: Conversions

Module Type: Package Specification

Module Purpose: This package ties all the generic conversion packages together and provides their common utilities and exceptions.

Module Description:

References:

Design Documents:
- Real-Time Monitor Requirements
  - Real-Time Monitor Design

User's Manual:
- RTM User's Manual

Testing and Validation:
- none

Notes:
- none

Modification History:
- 02Sep87 rlv created

Distribution and Copyright Notice:
- TBD

Disclaimer:
- "This work was sponsored by the Department of Defense. The views and conclusions contained in this document are solely those of the author(s) and should not be interpreted as representing official policies, either expressed or implied, of Carnegie Mellon University, the U.S. Air Force, the Department of Defense, or the U.S. Government."

pragma page:
with System;
-- Use the type "Address".
--

package Conversions is

--_signals_that_the_value_in_the_character_string_is_the_wrong_type
--_for_the_variable. This exception is shared among all the
--generic_conversion_packages.

Illegal_Value: exception;

pragma page;
Module Name: Convert_Integers

Module Type: Package Specification

Module Purpose: This is a package of generic utilities to convert binary bit strings into integer character strings and character strings into binary bit strings.

Module Description: This package contains two generic procedures used for converting from integer binary bit strings to integer character strings. The package is set up to operate in a two-CPU configuration, with the generic function target_conversion doing any needed translations from the target numeric representation into the host numeric representation.

References:
Design Documents:
- Real-Time Monitor Requirements
- Real-Time Monitor Design

User's Manual:
- RTM User's Manual

Testing and Validation:
none

Notes:
none

Modification History:
04Jun87 rlvs created

Distribution and Copyright Notice:
TBD

Disclaimer:
This work was sponsored by the Department of Defense. The views and conclusions contained in this document are solely those of the author(s) and should not be interpreted as representing official policies, either expressed or implied, of Carnegie Mellon University, the U.S. Air Force, the Department of Defense, or the U.S. Government.

pragma page;
generic
--
-- Default width of the generated character strings.
Width: Positive := 15;
--
-- Integer source_representation (this is the host machine's type)
type Source_Representation is range <>;
--
-- Low-level conversion routine needed to convert from the target
-- representation to the host representation of the source type
-- (referred to as source_representation)
with function Target_Conversion (Raw_Value: In System.Address)
return Source_Representation;

package Convert_Integers Is

    procedure Make_String (Raw_Value: In System.Address;
Field_Size: In Integer := Width;
Value: out String);

-- Description:
-- Make_string takes a binary bit string and converts it into
-- an integer character string.

procedure Make_Value (Raw_Value: In String;
Value: In System.Address);

-- Description:
-- Make_value takes an integer character string and converts it into a
-- binary bit string.

end Convert_Integers;

pragma page;
Module Name:
Convert_Floats

Module Type:
Package Specification

Module Purpose:
This is a package of generic utilities to convert binary bit strings into real character strings and character strings into binary bit strings.

Module Description:
This package contains two generic procedures used for converting from real binary bit strings to real character strings.
The package is set up to operate in a two-CPU configuration, with the generic function target_conversion doing any needed translations from the target numeric representation into the host numeric representation.

References:
Design Documents:
Real-Time Monitor Requirements
Real-Time Monitor Design

User's Manual:
RTM User's Manual

Testing and Validation:
none

Notes:
none

Modification History:
04Jun87 rvs created

Distribution and Copyright Notice:
TBD

Disclaimer:
"This work was sponsored by the Department of Defense.
The views and conclusions contained in this document are solely those of the author(s) and should not be interpreted as representing official policies, either expressed or implied, of Carnegie Mellon University, the U.S. Air Force, the Department of Defense, or the U.S. Government."
generic
-- Default width of the generated character strings.
--
Width: Positive = 15;
--
-- Integer type source, this is the host machine's type
--
type Source_Representation is digits <>;
--
-- Low-level conversion routine needed to convert from the target
-- representation to the host representation of the source type
-- (referred to as source_representation)
--
with function Target_Conversion (Raw_Value: In System.Address)
return Source_Representation;

package Convert_Floats is

procedure Make_String (Raw_Value: In System.Address;
Field_Size: In Integer := Width;
Value: out String);

procedure Make_Value (Raw_Value: In String;
Value: In System.Address);

end Convert_Floats.

pragma page

75
Module Name:
- Convert_Enumerations

Module Type:
- Package Specification

Module Purpose:
- This is a package of generic utilities to convert binary bit strings
  into enumeration character strings and character strings into
  binary bit strings.

Module Description:
- This package contains two generic procedures used for converting
  from enumeration binary bit strings to enumeration character strings.
  The package is set up to operate in a two-CPU configuration, with
  the generic function target_conversion doing any needed translations
  from the target numeric representation into the host numeric
  representation.

References:
- Design Documents:
  - Real-Time Monitor Requirements
  - Real-Time Monitor Design
- User's Manual:
  - RTM User's Manual

Testing and Validation:
- none

Notes:
- none

Modification History:
- 04Jun86 this module created

Distribution and Copyright Notice:
- 

Disclaimer:
- "This work was sponsored by the Department of Defense.
  The views and conclusions contained in this document are
  solely those of the authors and should not be interpreted as
  representing official policies, either expressed or implied.
  "Carnegie Mellon University, the U.S. Air Force,
  the Department of Defense, or the U.S. Government."

Pragma page
generic
  -- Default width of the generated character strings
  Width: Positive := 15;
  -- Integer type source, this is the host machine's type
  type Source_Representation is (<>);
  -- Low-level conversion routine needed to convert from the target
  -- representation to the host representation of the source type
  -- (referred to as source_representation)
  with function Target_Conversion (Raw_Value: In System.Address)
    return Source_Representation;

package ConvertEnumerations is

  procedure Make_String (Raw_Value: In System.Address;
      Field_Size: In Integer := Width;
      Value: out String);

  procedure Make_Value (Raw_Value: In String;
      Value: In System.Address);

end ConvertEnumerations;

pragma page
package Conversions is

package Convert Integers is separate

package Convert Floats is separate

package Convert Enumerations is separate

end Conversions;

pragma Page;
Module Name: Convert_Integers

Module Type: Package Body

Module Description:
This package contains two generic procedures used for converting from integer binary bit strings to integer character strings. It does this using the services of text_io and unchecked conversion.

References:
Design Documents: none

Testing and Validation: none

Notes: none

Modification History:
04Jun87 r1vs created

Distribution and Copyright Notice:
TBD

Disclaimer:
This work was sponsored by the Department of Defense. The views and conclusions contained in this document are solely those of the author(s) and should not be interpreted as representing official policies, either expressed or implied, of Carnegie Mellon University, the U.S. Air Force, the Department of Defense, or the U.S. Government.

pragma page.
with Text_lo;
-- Need services "put" and "get".

with Unchecked_Conversion;
-- Need service "unchecked_conversion".

package Convert_Integers is
-- Instantiate an io package to manipulate integer types.

package internal_lo is new Text_lo.Integer_lo (Source_Representation);
-- Create type and objects needed to access memory as integer values,
-- given system addresses as input.

  type Integer_Pointer is access Source_Representation;
  New_Integer_Value: Integer_Pointer.

function Address_To_Integer_Pointer is new Unchecked_Conversion
  (Source => System.Address,
   Target => Integer_Pointer);

function Integer_Pointer_To_Address is new Unchecked_Conversion
  (Source => Integer_Pointer,
   Target => System.Address);

pragma page:
procedure Make_String (Raw_Value in System.Address
Field_Size in Integer = Width
Value out String) is

-- Description:
-- Make_string takes a binary bit string and converts it into
-- an integer character string. It does this by using
-- target conversion to map the target bit representation of an
-- integer into the host version of an integer and then
-- uses text_to to convert the bits into an integer character string

-- Parameter Description:
-- raw_value -> The address of the binary bit string to be
-- converted
-- field_size -> The number of characters needed in the output
-- string
-- value -> The character image of the binary bit string as
-- an integer

-- Notes:
-- none

begin
  Internal To Put To => Value ; Field Size;
  Item => Target Conversion(Raw Value).
exception
  when Text_To Layout Error =>
    Value ; Field Size => ’ ’;
  when others =>
    RAISE;
end Make_String;

pragma page
procedure M1ia

Description
- M1ia is a procedure that is used in the context of a larger program.

Parameter Description
- M1ia takes an integer parameter.

Notes
- The procedure is used to handle exceptions in the program.

begin
- M1ia process

exception
- when exception
  RAISE
- when others
  RAISE
end M1ia
<table>
<thead>
<tr>
<th><strong>Module Name</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Module Type</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Module Description</strong></td>
<td></td>
</tr>
<tr>
<td><strong>References</strong></td>
<td>Design Documents</td>
</tr>
<tr>
<td><strong>Testing and Validation</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Notes</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Modification History</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Distribution and Copyright Notice</strong></td>
<td></td>
</tr>
</tbody>
</table>
with Text to
   Need services "put" and "get"

with Unchecked Conversion
   Need service "unchecked conversion"

package Convert Float IS

package Internal IS new Text to Float to (Source Representation)

create type and objects needed to access memory as float values
given system addresses as input

type Real Pointer IS access Source Representation
   new Real via Real Pointer

function Address to Real Pointer IS new Unchecked Conversion
   Source to System Address
   Target to Real Pointer

function Real Pointer to Address IS new Unchecked Conversion
   Source to Real Pointer
   Target to System Address

pragma
procedure Make_String (Raw_Value in System.Address;
   Field_Size in Integer := Width;
   Value out String) is

-- Description:
-- Make_string takes a binary bit string and converts it into
-- a real character string. It does this by using
-- target_conversion to map the target bit representation of a
-- real into the host version of a real and then
-- uses text_io to convert the bits into a real character string
--
-- Parameter Description:
-- raw_value -> The address of the binary bit string to be
-- converted
-- field_size -> The number of characters needed in the output
-- string
-- value -> The character image of the binary bit string as
-- a real.

-- Notes:
-- none

begin
   Interna_io.Put (To => Value(1 Field_Size),
                 Item => Target_Conversion(Raw_Value)),
exception
   when Text_io.Layout_Error =>
      Value(1 Field_Size = (1 Field_Size => '*'))
   when others =>
      RAISE
end Make_String

pragma page
procedure Make_Value (Raw_Value: In String;
Value: In System.Address) is

-- /Description:
-- / Make_Value takes a real character string and converts it
-- / into a binary bit string. It does this by converting
-- / the address where the data are to be stored into a pointer
-- / to a real and then uses text_io to get a real out of
-- / a string and store it at the pointer.
-- /
-- / Parameter Description:
-- / raw_value -> The character string to be converted.
-- / value -> The address where the resulting bit string is to be
-- / stored.
-- /
-- / Notes:
-- / none
-- /
Value_Location_Real_Pointer := Address_To_Real_Pointer(Value);
End_Of_Value := Integer;
begin
  Interna_io.Get (From => Raw_Value,
  Item => Value_Location.all,
  Last => End_Of_Value);
exception
  when Text_io.Data_Error =>
    RAISE _illegal_Value,
  when others =>
    RAISE;
end Make_Value;

end Convert_Floats;
pragma page
Module Name: Convert_Enumerations

Module Type: Package Body

Module Description:
This package contains two generic procedures used for converting:
from real binary bit strings to real character strings.
It does this using the services of text to and unchecked conversion.

References:

Design Documents:
none

Testing and Validation:
none

Notes:
none

Modification History:
04Jun87 files created

Distribution and Copyright Notice
TBD

Disclaimer:
"This work was sponsored by the Department of Defense.
The views and conclusions contained in this document are
solely those of the authors, and should not be interpreted as
representing official policies, either expressed or implied
of Carnegie Mellon University, the U.S. Air Force
the Department of Defense or the U.S. Government."
with "text":
  Need services "text" and "text"

with unchecked conversion
  Need services "Unchecked" and "text"

separate conversions

package Content Enumeration is

  -- Define package "content" for conversion

package Internal is new Textual Enumeration in Source Representation

Create type and objects needed to access memory representation value
given system addresses as input.

type Enum Pointer is access Source Representation
  New Enum Value: Enum Pointer.

function Addr to Enum Pointer is new Unchecked Conversion
  Source => System Address
  Target => Enum Pointer

function Enum Pointer to Addr is new Unchecked Conversion
  Source => Enum Pointer
  Target => System Address

pragma page
procedure Make_String(Raw_Value in System.Address, Field_Size in Integer = Width, Value out String) is

Description:
Make_String takes a binary bit string and converts it into an enumeration character string. It does this by using target conversion to map the target bit representation of an enumeration into the host version of an enumeration and then uses Text_IO to convert the bits into an enumeration character string.

Parameter Description:
- raw_value -> The address of the binary bit string to be converted.
- field_size -> The number of characters needed in the output string.
- value -> The character image of the binary bit string as an enumeration.

Notes:
- none

begin
  Internal_IO.Put (To => Value(1..Field_Size), Item => Target_Conversion(Raw_Value));
exception
  when Text_IO.Layout_Error =>
    Value(1..Field_Size) := (1..Field_Size => '*');
  when others =>
    RAISE;
end Make_String;

pragma page;
procedure Make_Value (Raw_Value: In String;
    Value: In System.Address) is
--/*********************************************************************************/

--/ Description:
--/ Make_value takes an enumeration character string and converts it
--/ into a binary bit string. It does this by converting
--/ the address where the data are to be stored into a pointer
--/ to an enumeration and then uses text_io to get an enumeration out of
--/ a string and store it at the pointer.
--/
--/ Parameter Description:
--/ raw_value -> The character string to be converted.
--/ value -> The address where the resulting bit string is to be
--/ stored.
--/
--/ Notes:
--/ none
--/*********************************************************************************/
    Value_Location: Enum_Pointer := Address_To_Enum_Pointer(Value);
    End_Of_Value: Integer;
begin
    Internal_Io.Get (From => Raw_Value,
        Item => Value_Location.all,
        Last => End_Of_Value);
exception
    when Text_Io_Data_Error =>
        RAISE Illegal_Value;
    when others =>
        RAISE;
end Make_Value;

end Convert_Enumerations;
pragma page;
Module Name:
Types_Manager

Module Type:
Package Specification

Module Purpose:
This package is the interface to all the underlying type representations used by the application.

Module Description:
This package contains all the knowledge in the system about types. It is both the database of legal (i.e., displayable) types and the mechanism by which data is converted from the internal RTM representation to a user-readable form.

References:
Design Documents:
- Real-Time Monitor Requirements
- Real-Time Monitor Design

User's Manual:
- RTM User's Manual

Testing and Validation:
none

Notes:
none

Modification History:
22May87 rvs created

Distribution and Copyright Notice:
TBD

Disclaimer:
"This work was sponsored by the Department of Defense. The views and conclusions contained in this document are solely those of the author(s) and should not be interpreted as representing official policies, either expressed or implied, of Carnegie Mellon University, the U.S. Air Force, the Department of Defense, or the U.S. Government."

pragma page;
with System;
-- Use the type "address".
--

package Types_Manager is
--
-- Type identifier, used externally to refer to a named type.
--
type Valid_Rtm_Type is private;

procedure Convert_Value_To_String (Data_Type: in Valid_Rtm_Type;
    Raw_Data: in System.Address;
    Number_Of_Characters: in Integer;
    The_Value: out String);

--/ Description:
--/ This module converts from the internal representation used
--/ by the RTM in storing variable values into strings that
--/ are displayable to the user.
--/
--/ Parameter Description:
--/  data_type -> The Ada data type of raw data.
--/  raw_data -> The address of the binary bit string to convert.
--/  number_of_characters -> The number of characters needed in the
--/  value string.
--/  the_value -> A string containing the displayable value.
--/-----------------------------------------------

procedure Convert_String_To_Value (Data_Type: in Valid_Rtm_Type;
    Raw_Data: in System.Address;
    The_Value: in String);

--/ Description:
--/ This module converts from the string entered by the user
--/ into the internal representation used by the RTM.
--/
--/ Parameter Description:
--/  data_type -> The Ada data type of raw data.
--/  raw_data -> The address of the binary bit string to convert.
--/  the_value -> The string whose value the user wishes deposited into
--/ application memory.
--/-----------------------------------------------

function Find (Name: in String) return Valid_Rtm_Type;

--/ Description:
--/ This module is the lookup entry used to locate legal types
--/ It maps data obtained from the library_interface into types
--/ which the types_manager can convert.
--/
--/ Parameter Description:
--/  name -> The name of the Ada type associated with
--/  a variable.
--/  return -> The internal Identifier used to refer
--/  to the type.

83 October 1987
procedure GetTypeInformation (Type_Identifier: In Valid_Rtm_Type;
                         Type_Length: out Integer;
                         Indirection_Indicator: out Boolean);

-- Description:
-- This module takes a type identifier and returns detailed
-- information about the structure of the type to the caller.

-- Parameter Description:
-- type_identifier -> Identifier of the type about which
-- information is needed.
-- type_length -> The size of the underlying type in the
-- size of the storage units used by the RTM
-- (i.e., smallest_units).
-- indirection_indicator -> Logical flag which when
-- true => an access type
-- false => any other type

-- Exceptions
-- TypeNotFound: exception;

-- Exception used to signal illegal value string for type.
-- IllegalValue: exception;

private

    type Valid_Rtm_Type is new Integer;

end Types_Manager;
pragma page;
Module Name:
Types_Manager

Module Type:
Package Body

Module Description:
This package embodies type database and the operations needed
to access the database and convert data from bit strings into
canacter strings.

The type database consists of an array of records (defined
below), that contains all the data needed to convert a
bit string into a value. All data types accessible to the
user must be defined in this database and have corresponding
entries in the low-level conversion routines for their
implementation.

References:
Design Documents:
Real-Time Monitor Requirements
Real-Time Monitor Design

User's Manual:
RTM User's Manual

Testing and Validation:
none

Notes:
none

Modification History:
04Jun87 rvs created

Distribution and Copyright Notice:
TBD

Disclaimer:
"This work was sponsored by the Department of Defense.
The views and conclusions contained in this document are
solely those of the author(s) and should not be interpreted as
representing official policies, either expressed or implied,
of Carnegie Mellon University, the U.S. Air Force,
the Department of Defense, or the U.S. Government."
with Test_Strip;

-- Need access to the type definitions defined in the dummy application.

with Conversions;

-- Use the generic packages: "convert_floats", "convert_integers" and
-- "convert_enumerations".
-- Use the exception "illegal-value".

with Unchecked_Conversion;
-- Use service "unchecked_conversion".

package Types_Manager is

-- Define the names of all the legal types

type Valid_Type_Name is (Integers, Floats, Rtm_Enum1, Rtm_Record);

-- Define all the data needed about each type:
-- type_name_as_string -> A character string version of the type name.
-- This must match exactly with the type as it
-- exists in the application program, where the
-- type_name is a convenient enumeration literal
-- for this type.

-- type_name -> An enumeration literal for the type.

-- type_length -> The size of the type in smallest units.

-- display_width -> Number of characters needed to display a value
-- of the type type_name.

-- indirection_level -> An integer that indicates how many levels of
-- indirect access the type represents.

type Type_Representation is record
  Type_Name_As_String: String(1..256);
  Type_Name: Valid_Type_Name;
  Type_Length: Integer := 0;
  Display_Width: Integer := 25;
  Indirection_Level: Integer := 0;
end record;

-- Define the table that holds all the type information; define
-- type_name_as_string in body.

Number_Of_Valid_Types: Valid_Rtm_Type :=
  Valid_Type_Name'Pos(Valid_Type_Name'Last);
Valid_Rtm_Types: array (0..Number_Of_Valid_Types) of Type_Representation :=
((Type_Name_As_String => (others => ' '),
  Type_Name => Integers, Type_Length => 1,
  Display_Width => 10, Indirection_Level => 0),
((Type_Name_As_String => (others => ' '),
  Type_Name => Floats, Type_Length => 1,
  Display_Width => 10, Indirection_Level => 0),
((Type_Name_As_String => (others => ' '),
  Type_Name => Rtm_Enum1, Type_Length => 1,
  Display_Width => 5, Indirection_Level => 0),
((Type_Name_As_String => (others => ' '),
  Type_Name => Rtm_Record, Type_Length => 1,
  Display_Width => 5, Indirection_Level => 0),
...
(Type_Name_As_String => (others => ' '),
Type_Name => Rtm_Record, Type_Length => 2,
Display_Width => 20, Indirection_Level => 0));

type Float_Pointer is access Float;
function Address_To_Float_Pointer is new Unchecked_Conversion
  (Source => System.Address,
   Target => Float_Pointer);

type Integer_Pointer is access Integer;
function Address_To_Integer_Pointer is new Unchecked_Conversion
  (Source => System.Address,
   Target => Integer_Pointer);

type Rtm_Enum_Pointer is access TestStub.Rtm_Enum;
function Address_To_Rtm_Enum_Pointer is new Unchecked_Conversion
  (Source => System.Address,
   Target => Rtm_Enum_Pointer);

pragma page;
function Default_Float_Conversion (Raw_Value: In System.Address)
  return Float is
  -- */ Description:
  -- */ Convert from a bit string at a system address to a floating
  -- */ point value. This is valid for a one-CPU configuration
  -- */ only.
  -- */
  -- */ Parameter Description:
  -- */ raw_value -> The address of the bit string to convert.
  -- */
  -- */ Notes:
  -- */ none
  -- */
  Value_Pointer: Float_Pointer;
begin
  Value_Pointer := Address_To_Float_Pointer(Raw_Value);
  RETURN Value_Pointer.all;
end Default_Float_Conversion;
pragma Inline (Default_Float_Conversion);

package Rtm_Reals is new Conversions.Convert_Floats
  (Width => 15,
   Source_Representation => Float,
   Target_Conversion => Default_Float_Conversion);

pragma page;
function Default_Integer_Conversion (Raw_Value: In System.Address) 
  return Integer is 
  ...............................................
  --/ Description: 
  --/ Convert from a bit string at a system address to an integer 
  --/ value. This is valid for a one-CPU configuration 
  --/ only. 
  --/ 
  --/ Parameter Description: 
  --/ raw_value -> The address of the bit string to convert. 
  --/ 
  --/ Notes: 
  --/ none 
  ...............................................
  Value_Pointer: Integer_Pointer;
  begin
    Value_Pointer := Address_To_Integer_Pointer(Raw_Value);
    RETURN Value_Pointer.all ;
  end Default_Integer_Conversion;
pragma Inline (Default_Integer_Conversion);
  --
  -- Create the package to convert from bit strings to integers.
  --

package Rtm_Integers is new Conversions.Convert_Integers 
  (Width => 15, 
   Source_Representation => Integer, 
   Target_Conversion => Default_Integer_Conversion);
pragma page;
function Rtm_Enum_Conversion (Raw_Value: In System.Address) 
  return Test_Stub.Rtm_Enum is 
  -- /*-----------------------------------------------
  -- Description:
  -- Convert from a bit string at a system address to an
  -- enumeration value. This is valid for a one-CPU configuration
  -- only.
  --
  -- Parameter Description:
  -- raw_value -> The address of the bit string to convert.
  --
  -- Notes:
  -- none
  -- /*-----------------------------------------------
    Value_Pointer: Rtm_Enum_Pointer,
    begin
      Value_Pointer := Address_To_Rtm_Enum_Pointer(Raw_Value);
      RETURN Value_Pointer.all;
    end Rtm_Enum_Conversion;
  pragma inline (Rtm_Enum_Conversion);
  --
  -- Create the package to convert from bit strings to rtm_enum enumerations.
  --
  package Rtm_Enums is new Conversions.ConvertEnumerations
    (Width => 5,
     Source_Representation => Test_Stub.Rtm_Enum,
     Target_Conversion => Rtm_Enum_Conversion);
  pragma page;
Visible procedures

procedure Convert_Value_To_String (Data_Type: in Valid_Rtm_Type;
    Raw_Data: in System.Address;
    Number_Of_Characters: in Integer;
    The_Value: out String) is separate;

procedure ConvertString_To_Value (Data_Type: in Valid_Rtm_Type;
    Raw_Data: in System.Address;
    The_Value: in String) is separate;

function Find (Name: in String) return Valid_Rtm_Type is separate;

procedure Get_Type_Information (Type_Identifier: in Valid_Rtm_Type;
    Type_Length: out Integer;
    Indirection_Indicator: out Boolean) is separate;

Package Body

The body is responsible for initializing the string versions of all the type names.

begin
    Valid_Rtm_Types(0).TypeName_As_String(1..7) := "integer";
    Valid_Rtm_Types(1).TypeName_As_String(1..5) := "float";
    Valid_Rtm_Types(2).TypeName_As_String(1..9) := "rtm_enum1";
    Valid_Rtm_Types(3).TypeName_As_String(1..10) := "rtm_record";
end Types_Manager;

pragma page;
separate (Types_Manager)

procedure Convert_Value_To_String (Data_Type: in Valid_Rtm_Type;
   Raw_Data: in System.Address;
   Number_Of_Characters: in Integer;
   The_Value: out String) is

   -- *******************************************************************************
   -- Description:
   -- This module converts from the internal representation used
   -- by the RTM in storing variable values into strings which
   -- are displayable to the user. Since the bit pattern in the
   -- internal representation (collected by the Rtm_Core) may or
   -- may not have an analog in the machine running the user interface,
   -- a package of conversion routines is used to translate the bits into
   -- a form the host machine can handle. This procedure then takes
   -- the bits and forms a user-readable string.
   --
   -- Parameter Description:
   -- data_type -> The Ada data type of raw data.
   -- raw_data  -> The address of the binary bit string to convert.
   -- number_of_characters -> The number of characters needed in the
   -- value string.
   -- the_value -> A string containing the displayable value.
   --
   -- Notes:
   -- none
   --*******************************************************************************

   begin
     The_Value := (The_Value'range => '.');
     case Valid_Rtm_Types(Data_Type).Type_Name is
     when Integers =>
       Rtm_Integers.Make_String (Raw_Value => Raw_Data,
                                Field_Size => Number_Of_Characters,
                                Value => The_Value);
     when Floats =>
       Rtm_Reals.MakeString (Raw_Value => Raw_Data,
                             Field_Size => Number_Of_Characters,
                             Value => The_Value);
     when Rtm_Enum1 =>
       RtmEnums.MakeString (Raw_Value => Raw_Data,
                            Field_Size => Number_Of_Characters,
                            Value => The_Value);
     when Rtm_Record =>
       null;
     when others =>
       null;
     end case;
   end Convert_Value_To_String;

pragma page;
procedure Convert_String_To_Value (Data_Type: in Valid_Rtm_Type;
   Raw_Data: in System.Address;
   The_Value: in String) is

--/ Description:
--/ This module converts from the string entered by the user
--/ into the internal representation used by the RTM.
--/ Since the bit pattern in the internal representation
--/ (collected by the Rtm_Core), may or may not have an analog
--/ in the machine running the user interface, a package of
--/ conversion routines is used to translate the bits into a
--/ form the target machine can handle.
--/
--/ Parameter Description:
--/ data_type  -> The Ada data type of raw data.
--/ raw_data   -> The address of the binary bit string to convert.
--/ the_value  -> The string whose value the user wishes deposited into
--/ application memory.
--/
--/ Notes:
--/ none
--/........................................................................
begin
   case Valid_Rtm_Types(Data_Type).Type_Name is
      when Integers =>
         Rtm_Integers.Make_Value (The_Value, Raw_Data);
      when Floats =>
         Rtm_Reals.Make_Value (The_Value, Raw_Data);
      when Rtm_Enum1 =>
         Rtm_Enums.Make_Value (The_Value, Raw_Data);
      when Rtm_Record =>
         null;
      when others =>
         null;
   end case;
exception
   when Conversions.Illegal_Value =>
      RAISE Illegal_Value;
   when others =>
      RAISE;
end Convert_String_To_Value;
pragma page;
with CaseInsensitiveStringComparison;
-- Use service "equal";

separate (Types_Manager)
function Find (Name: in String) return Valid_Rtm_Type is

-- Description:
-- This module is the lookup entry used to locate legal types.
-- It maps data obtained from the library_interface into types
-- that the types_manager can convert.
--
-- Parameter Description:
-- name -> The name of the Ada type associated with
-- a variable.
-- return -> The internal identifier used to refer
-- to the type.
--
-- Notes:
-- When a type is not found in the type database,
-- the exception "type_not_found" is raised. Given that
-- all this information is coming from the same source, this
-- exception should never be used.
--
package Cisc renames CaseInsensitiveStringComparison;
Type_Location: Valid_Rtm_Type;
begin

-- Loop through the types in the database until we find the type or
-- run out of database.

for Type_Location in 0..NumberOf_Valid_Types loop
    if Cisc.Equal (Name,
                   Valid_Rtm_Types (Type_Location).Type_Name_Asc (Name'range))
    then
        RETURN Type_Location;
    end if;
end loop;
RAISE Type_Not_Found;
end Find;
pragma page;
Separate (Types_Manager)

procedure Get_Type_Information (Type_Identifier: in Valid_Rtm_Type;
   Type_Length: out Integer;
   Indirection_Indicator: out Boolean) is

--/ Description:
--| This module takes a type identifier and returns detailed
--| information about the structure of the type to the caller.
--/
--/ Parameter Description:
--| type_identifier  -> Identifier of the type about which
--|       information is needed.
--| type_length  -> The size of the underlying type in the
--|        size of the storage units used by the RTM
--|              (i.e., smallest_units).
--| indirection_indicator  -> Logical flag which when
--|        true => an access type
--|        false => any other type
--/******************************************************/

begin

-- Extract the length (in smallest_units) of the type.

   Type_Length := Valid_Rtm_Types(Type_Identifier).Type_Length;

-- Determine if the type is an access pointer, based on its level
-- of indirection, i.e., 0 => no indirection.

   if Valid_Rtm_Types(Type_Identifier).Indirection_Level = 0 then
      Indirection_Indicator := False;
   else
      Indirection_Indicator := True;
   end if;

end Get_Type_Information;

pragma page;

105 October 1987
Module Name:
Variable_Database

Module Type:
Package Specification

Module Purpose:
This module manages the interface to the underlying variable database.

Module Description:
This module manages the variable database created out of information obtained via the library_interface. The variable_database holds all the variables accessible to the user. This package hides all details about the structure and manipulation of the variable database. It also knows how to initialize the database at startup time.

This module manages the structure that is the variable_database; the actual data comes from the Library_interface, which is responsible for the fidelity and content of the variable_database. The Variable_Database package has all the data we need about a variable (name, base_address, type_name, and type_identifier).

References:
Design Documents:
- Real-Time Monitor Requirements
- Real-Time Monitor Design

User's Manual:
- RTM User's Manual

Testing and Validation:
- none

Notes:
- none

Modification History:
08Jul87 rlv created

Distribution and Copyright Notice:
TBD

Disclaimer:
"This work was sponsored by the Department of Defense. The views and conclusions contained in this document are solely those of the author(s) and should not be interpreted as representing official policies, either expressed or implied, of Carnegie Mellon University, the U.S. Air Force, the Department of Defense, or the U.S. Government."

-------------------------------
pragma page;
with Library_Interface;
-- Use type "variable_representation".
--

package Variable_Database is

    type The_Variable Is access Library_Interface.Variable_Representation;

    procedure Initialize_Database;
    --/ Description:
    --/ This module is responsible for building the variable database
    --/ by whatever means are available.
    --/
    --/ Parameter Description:
    --/ none
    --/*********************************************************/

    function Find (Name: In String) return The_Variable;
    --/ Description:
    --/ This function searches the variable database for the variable
    --/ passed in.
    --/
    --/ Parameter Description:
    --/ name  -> The name of variable to look up.
    --/
    --/ Notes:
    --/ When a variable is not found in the variable database,
    --/ the exception "variable_not_found" is raised.
    --/*********************************************************/

    Variable_Not_Found: exception;

end Variable_Database;
pragma page;
```plaintext
-- Module Name:
--  Variable_Database

-- Module Type:
--  Package Body

-- Module Description:
-- This module encapsulates the actual structure of the variable
database. The database is stored as an ordered binary tree.

-- References:
-- Design Documents:
--  Real-Time Monitor Requirements
--  Real-Time Monitor Design

-- User's Manual:
--  RTM User's Manual

-- Testing and Validation:
-- none

-- Notes:
-- none

-- Modification History:
-- 16Apr87 rvs created

-- Distribution and Copyright Notice:
-- TBD

-- Disclaimer:
-- "This work was sponsored by the Department of Defense.
The views and conclusions contained in this document are
solely those of the author(s) and should not be interpreted as
representing official policies, either expressed or implied,
of Carnegie Mellon University, the U.S. Air Force,
the Department of Defense, or the U.S. Government."
```

Pragma page;
with CaseInsensitive_String_Comparison;
-- Use the services "less" and "equal".
--
with Binarytrees;
-- Use generic package "binarytrees".
-- Use type "tree".
-- Use service "create".
--
package Variable_Database is
--
-- Set up a shorthand notation for the string package.
--
package Cisc renames CaseInsensitive_String_Comparison;
pragma page;
-- Internal procedures

-- Define the ordering function to be used by the tree package, and create a package to manipulate trees of pointers.

function Ordering (Left: The_Variable; Right: The_Variable) return Boolean;

package Db is new Binarytrees (Itemtype => The_Variable, "<" => Ordering);

-- Create the variable database.

Variable_Database: Db.Tree := Db.Create;

function Ordering (Left: The_Variable; Right: The_Variable) return Boolean Is

--/ Description:
-- This defines the ordering relation on the variable database. Since the elements in the tree are pointers, and we want the tree to be ordered alphabetically by variable name, we define the ordering function to use the pointers stored in the tree and access the name component of the record.

--/ Parameter Description:
-- left -> left child of the parent node.
-- right -> right child of the parent node.
-- return -> true, if the left child's name is less than the right child's name.
-- false, otherwise.

--/ Notes:
-- none

begin
  RETURN Cisc.Less (Left.Variable_Name, Right.Variable_Name);
end;

-- Visible procedures
--
procedure Initialize_Database Is separate;

pragma page;
function Find (Name: In String) return The_Variable Is

--|--------------------------------------------------------------------------------|
--| Description:
--| This function searches the variable database for the variable passed in. It does this by making a tree iterator and walking the binary tree. Since the tree is ordered, this amounts to a binary search of the tree.
--| Parameter Description:
--| name -> The name of variable to look up.
--| Notes:
--| When a variable is not found in the variable database, the exception "variable_not_found" is raised.

Database_Position := Db.Treeiter;
Variable_Location := The_Variable;
begin
--
-- We locate the variable of interest by iterating through all the variables in the database until find it or run out of tree.
--
Database_Position := Db.Treeiter(Variable_Database);
while Db.More(Database_Position) loop
  Db.Next(Database_Position, Variable_Location);
  if Cisc.Equal(Variable_Location.Variable_Name(Name'range), Name) then
    RETURN Variable_Location;
  end if;
end loop;
RAISE Variable_Not_Found;
end Find;

end Variable_Database;
pragma page;
with Unchecked_Deallocation;
   -- Use the service "unchecked_deallocation".

with Types_Manager;
   -- Use the service "find".

separate (Variable_Database)
procedure Initialize_Database is
   description:
   -- This module is responsible for building the variable database
   -- by whatever means are available.
   --
   -- parameter description:
   -- none
   --
   -- notes:
   -- all of the system-dependent issues related to obtaining
   -- object addresses have to be isolated in these packages:
   -- Library_interface: for static data information.
   -- Address_generator: for dynamic data information.
   -- these are the packages that must be changed to reflect the
   -- system configuration and environment.
   -----------------------------------------------
   procedure Free is new Unchecked_Deallocation
      (Library_interface.Variable_Representation,The_Variable);
      Variable_Position: Library_interface.Variable_iterator;
      Node_Root: Db.Tree;
      Found_Variable: Boolean;
      The_Next_Variable: The_Variable;

   begin
   --
   -- the basic operation is same for all the variables:
   -- build a variable_representation record.
   -- insert the record into the tree.
   -- repeat for all variables.
   --
      Library_interface.Make_iterator(Variable_Position);
      while Library_interface.More(Variable_Position) loop
         begin
            The_Next_Variable := new Library_interface.Variable_Representation;
            Library_interface.Get_next
               (The_iterator => Variable_Position,
                Variable_information => The_Next_Variable.all);
            The_Next_Variable.Data_Type := Types_Manager.Find
               (Name => The_Next_Variable.Variable_Type);
            Db.Insertnode(N => The_Next_Variable,
                           T => Variable_Database,
                           Root => Node_Root,
                           Exits => Found_Variable);
            exception
               when Types_Manager.Type_Not_Found =>
                  Free (The_Next_Variable);
            end;

113
October 1987
end loop;
end Initialize(Database);
pragma page;
Module Name: Library Interface

Module Type: Package Specification

Module Purpose: This module provides the interface needed by the RTM to build the variable database.

Module Description: This package presumes an interface into a compiler library mechanism that is capable of generating an address for any statically allocated variable. The interface is extremely simple; it consists of three parts:

- make_iterator: Initializes an iteration object and allows the caller to step through the library structure without regard to its organization.
- get_next: Returns all the relevant information about the next variable.
- more: Signals when the entire structure has been traversed, and there are no more variables.

References:
- Design Documents:
  - Real-Time Monitor Requirements
  - Real-Time Monitor Design

- User's Manual:
  - RTM User's Manual

- Testing and Validation:
  - none

Notes:
- none

Modification History:
- 02Jun87 nvs created

Distribution and Copyright Notice:
- TBD

Disclaimer:
- "This work was sponsored by the Department of Defense. The views and conclusions contained in this document are solely those of the author(s) and should not be interpreted as representing official policies, either expressed or implied, of Carnegie Mellon University, the U.S. Air Force,"
the Department of Defense, or the U.S. Government.*
with Types_Manager;
-- Use type "type_identifier".

package Library_Interface is
--
-- The iteration variable, used to control iteration.
--
type Variable_Iterator is private;
--
-- The information stored for a variable in the database is:
-- variable_name  -> Full Ada path name.
-- base_address   -> Memory address of data (in application memory).
-- variable_type  -> The name of the type.
-- data_type      -> The type identifier, used by the types_manager to
--                   refer to a type.
--
type Variable_Representation is record
  Variable_Name: String(1..100) := (others => ' ');
  Base_Address: Integer := 0;
  Variable_Type: String(1..100) := (others => ' ');
  Data_Type: Types_Manager.Valid_Rtm_Type;
end record;

pragma page;
procedure Make_Iterator (The_iterator: in out Variable_Iterator);
-- Description:
-- Make_Iterator initializes an iteration parameter to the start of
-- the library structure; this parameter is then used to
-- retrieve information from the structure.
--
-- Parameter Description:
-- the_iterator -> The iteration parameter used by the
-- caller to access the next item.

procedure GetNext (The_iterator: in out Variable_Iterator;
                     Variable_Information: out Variable_Representation);

function More (The_iterator: in Variable_Iterator) return Boolean;

-- Description:
-- More takes an iteration parameter and determines if there are
-- any additional variables yet to be processed.
--
-- Parameter Description:
-- the_iterator -> The iteration parameter used by the
-- caller to access the next item.
-- return -> true => there are more variables
--           false => The entire structure has been traversed

private
  type Variable_Iterator is new Integer;

end Library_Interface;
pragma page;
Module Name:  
Library_interface

Module Type:  
Package Body

Module Description:
This package presumes an interface into a compiler library 
mechanism that is capable of generating an address for any 
statically allocated variable. This interface is a dummy 
package that simply returns the addresses of the static 
data defined in the package test_stub. It has to be replaced 
by whatever mechanism is available on the target machine.

References:
Design Documents:  
Real-Time Monitor Requirements  
Real-Time Monitor Design

User's Manual:  
RTM User's Manual

Testing and Validation:
none

Notes:
none

Modification History:
02Jun87 rivs created

Distribution and Copyright Notice:
TBD

Disclaimer:
"This work was sponsored by the Department of Defense. 
The views and conclusions contained in this document are 
solely those of the author(s) and should not be interpreted as 
representing official policies, either expressed or implied, 
of Carnegie Mellon University, the U.S. Air Force, 
the Department of Defense, or the U.S. Government."

pragma page;
with System;
-- Use type "address".
--
with Unchecked_Conversion;
-- Use service "unchecked_conversion".
--
with Test_Stub;
-- Use data objects defined here for testing the monitor.
--

package Library_Interface is
--
-- Used to convert all the system addresses into integers so that
-- they can be stored in the variable database. In a system where
-- an address map is used, this routine will need to be reimplemented.
--
function Get_Address is new Unchecked_Conversion
  (Source => System.Address,
   Target => Integer);

pragma page;
procedure Make_iterator (The_iterator: in out Variable_iterator) is

-- Description:
-- Make_iterator initializes an iteration parameter to the start
-- of the library structure; this parameter is then used to
-- retrieve information from the structure.
--
-- Parameter Description:
-- the_iterator -> The iteration parameter used by the
-- caller to access the next item.
--
-- Notes:
-- none
--
begin
    The_iterator := 0;
end Make_iterator;

pragma page;
procedure Get_Next (The_Iterator in out Variable_Iterator;
     Variable_Information in out Variable_Representation) is

begin
    case The_Iterator is
        when 0 =>
            Variable_Information.Variable_Name(1..20) := "test_stub.my_integer";
            Variable_Information.Base_Address := Get_Address(TestStub.My_Integer'Address);
            Variable_Information.Variable_Type(1..7) := "integer";
        when 1 =>
            Variable_Information.Variable_Name(1..17) := "test_stub.my_real";
            Variable_Information.Base_Address := Get_Address(TestStub.My_Real'Address);
            Variable_Information.Variable_Type(1..5) := "float";
        when 2 =>
            Variable_Information.Variable_Name(1..17) := "test_stub.my_enum";
            Variable_Information.Base_Address := Get_Address(TestStub.My_Enum'Address);
            Variable_Information.Variable_Type(1..9) := "rtm_enum1";
        when 3 =>
            Variable_Information.Variable_Name(1..15) := "test_stub.int_2";
            Variable_Information.Base_Address := Get_Address(TestStub.Int_2'Address);
            Variable_Information.Variable_Type(1..7) := "integer";
        when 4 =>
            Variable_Information.Variable_Name(1..20) := "test_stub.my_pointer";
            Variable_Information.Base_Address := Get_Address(TestStub.My_Pointer'Address);
            Variable_Information.Variable_Type(1..10) := "rtm_record";
        when 5 =>
            Variable_Information.Variable_Name(1..22) := "test_stub.my_pointer.l";
            Variable_Information.Base_Address := Get_Address(TestStub.My_Pointer.l'Address);
            Variable_Information.Variable_Type(1..7) := "integer";
        when 6 =>
            Variable_Information.Variable_Name(1..22) := "test_stub.my_pointer.r";
            Variable_Information.Base_Address := Get_Address(TestStub.My_Pointer.r'Address);
            Variable_Information.Variable_Type(1..5) := "float";
        when 7 =>
            Variable_Information.Variable_Name(1..18) := "test_stub.my_array";
            Variable_Information.Base_Address := Get_Address(TestStub.My_Array'Address);
            Variable_Information.Variable_Type(1..8) := "array_10";
        when 8 =>
            Variable_Information.Variable_Name(1..21) := "test_stub.my_array(2)";
            Variable_Information.Base_Address := Get_Address(TestStub.My_Array(2)'Address);
            Variable_Information.Variable_Type(1..7) := "integer";
        when others =>
            null;
    end case;
end
end case;
The_Iterator := The_Iterator + 1;
end Get_Next;

pragma page;
function More (The_Iterator: In Variable_Iterator) return Boolean is

--/ Description:
--/ More takes an iteration parameter and determines if there are
--/ any additional variables yet to be processed.

--/ Parameter Description:
--/ the_iterator  -> The iteration parameter used by the
--/ caller to access the next item.
--/ return       > true  ==> there are more variables
--/           false  ==> the entire structure has been traversed

--/ Notes:
--/ none

begin
  if The_Iterator <= 8 then
    RETURN True;
  else
    RETURN False;
  end if;
end More;
end Library_Interface;
pragma page;
Module Name:
Address_Generator

Module Type:
Package Specification

Module Purpose:
Defines the address abstraction used to refer to physical addresses in application memory and provides the interface needed to compute the address.

Module Description:
This module hides all the details surrounding the generation of object address by presenting one uniform interface to the rest of the RTM.

References:
Design Documents:
Real-Time Monitor Requirements
Real-Time Monitor Design

User's Manual:
RTM User's Manual

Testing and Validation:
one

Notes:
one

Modification History:
04Aug87 rivos created

Distribution and Copyright Notice:
TBD

Disclaimer:
"This work was sponsored by the Department of Defense. The views and conclusions contained in this document are solely those of the author(s) and should not be interpreted as representing official policies, either expressed or implied, of Carnegie Mellon University, the U.S. Air Force, the Department of Defense, or the U.S. Government."

pragma page;
with Variable_Database;

package Address_Generator is

-- Address abstraction:
-- base_address  --> Static base of the object.
-- address_offset --> Offset from the base address of object
-- (for components of compound objects).
-- indirection -> Boolean marker for identifying access type objects:
--   true  => access type
--   false => any other (non-access) type

type Address_Representation is record
  Base_Address: Integer;
  Address_Offset: Integer;
  Indirection: Boolean;
end record;

-- Default address

Null_Address: Address_Representation := (0,0,False);

function Compute_Address (Variable_Name: In String)
return Address_Representation;

-- Description:
-- This module takes the database identifier of a variable and
-- computes the address of the variable.
--
-- Parameter Description:
-- the_variable   --> Name of variable for which address is needed.
-- return         --> Computed address of the variable.

end Address_Generator;
pragma page;
Module Name: Address_Generator
Module Type: Package Body

Module Description:
This module is responsible for implementing the address computation needed by the RTM. Currently, this is totally embodied by the Compute_Address procedure.

References:
Design Documents:
Real-Time Monitor Requirements
Real-Time Monitor Design

User's Manual:
RTM User's Manual

Testing and Validation:
none

Notes:
none

Modification History:
08Aug87 rvS created

Distribution and Copyright Notice:
TBD

Disclaimer:
"This work was sponsored by the Department of Defense. The views and conclusions contained in this document are solely those of the author(s) and should not be interpreted as representing official policies, either expressed or implied, of Carnegie Mellon University, the U.S. Air Force, the Department of Defense, or the U.S. Government."

with Types_Manager;
-- Use the service "find".

package Address_Generator is

function Compute_Address (Variable_Name: In String)
return Address_Representation Is separate;

end Address_Generator;
pragma page;
separate (Address_Generator)

function Compute_Address (Variable_Name: In String) return Address_Representation is

--/ Description:
--/   This module takes the database identifier of a variable and
--/   computes the address of the variable.
--/
--/ Parameter Description:
--/   the_variable -> Name of variable for which address is needed.
--/   return   -> Computed address of the variable.
--/
--/ Notes:
--/   No address offset is computed since all accessible variables are
--/   in the database and the base_address already has the offset
--/   taken into account.
--/

The_Variable: Variable_Database. The_Variable;
Address: Address_Generator. Address_Representation := Null_Address;
Address_Offset: constant Integer := 0;
Data_Length: Integer;
Access_Flag: Boolean;

begin
  The_Variable := Variable_Database. Find (Variable_Name);
  Types_Manager. Get_Type_Information (The_Variable. Data_Type,
                                          Data_Length,
                                          Access_Flag);
  Address := (The_Variable. Base_Address,
              Address_Offset,
              Access_Flag);
  RETURN Address;

exception
  when Variable_Database. Variable_Not_Found =>
    RAISE;
  when Types_Manager. Type_Not_Found =>
    RAISE;
  when others =>
    null;
end Compute_Address;
pragma page;
Module Name: Dialogue_manager

Module Type: Package Specification

Module Purpose: This package manages the interface to the application. It does this by hiding all details about retrieving variable information from the application.

Module Description: The Dialogue Manager manages the interface to the application in a number of ways:
1. It knows how to talk to the Rtm_Core, which is the RTM's interface to the application.
2. It knows how to convert data retrieved from the Rtm_Core into strings which the user can understand.

References:
Design Documents:
- Real-Time Monitor Requirements
- Real-Time Monitor Design

User's Manual:
- RTM User's Manual

Testing and Validation:
- none

Notes:
- none

Modification History:
- 20Apr87 rvvs created

Distribution and Copyright Notice:
- TBD

Disclaimer:
- "This work was sponsored by the Department of Defense.
- The views and conclusions contained in this document are solely those of the author(s) and should not be interpreted as representing official policies, either expressed or implied, of Carnegie Mellon University, the U.S. Air Force, the Department of Defense, or the U.S. Government."

pragma page;
with Calendar; use Calendar;
- The "time" and "duration" types are used.

package Dialogue_Manager is

- Internal (RTM) representation of a variable.
  type Variable_Identifier is private;

- All value functions return items of this type.
  subtype Value_String is String(1..80);

- The direction flag informs the Dialogue_Manager about the direction of
  access on a variable:
  read => Get the data from the application.
  write => Put the data in the application.
  type io_Usage is (Read,Write);

pragma page;
function Get_Value (Vid: in Variable_Identifier;
   Number_Of_Characters: in Integer := 80)
   return Value_String;

-- Description:
-- Converts the current value of a variable (denoted as a
-- variable identifier) into a character string containing
-- the requested number of characters. Since this operation
-- requires a variable_identifier, it can only be performed on an
-- active variable. This function returns
-- a string appropriate to the type of the variable.
--
-- Parameter Description:
-- vid -> The identifier of the variable whose value is needed
-- (obtained from get_identifier above).
-- number_of_characters -> The number of characters needed in the
-- value string.
-- return -> A string containing a displayable value. For a
-- composite structure, the individual components
-- are delimited by the separator selected.
--
pragma page;
function Get_Time_Of_Value (Vid: In Variable_identifier)
    return Value_String;

-- Description: 
-- Convert the time associated with a variable value (i.e., the 
-- collection time) into a displayable string. Since this operation 
-- requires a variable_identifier, it can only be performed on an 
-- active variable.

-- Parameter Description:
-- vid -> The identifier of the variable whose time is needed 
-- (obtained from get_identifier above).
-- return -> The collection time of the current value as a 
-- displayable string.

pragma page;
procedure Set_Value (Vid: in Variable_Identifier;
Value: in Value_String);

--/ Description:
--/ Sets the value of the selected variable, normally in preparation for
--/ a set (write) operation. Since this operation requires a
--/ variable_identifier, it can only be performed on an active
--/ variable.
--/
--/ Parameter Description:
--/ vid -> The identifier of the variable whose value is being set
--/ (obtained from get_identifier above).
--/ value -> A string containing the value to deposit into
--/ the variable internal representation. For composite
--/ structure, the separator selected.
--/

pragma page;
function Activate (Name: In String;
   Rate: In Duration;
   Starting_Time: Time;
   Usage: In Io_Usage) return Variable_Identifier;

Description:
This entry activates variables for data collection or modification.
It keeps track of which variables in the variable database are of
interest to the user.

Parameter Description:
vid -> The identifier of the variable to activate
( obtained from get_identifier above).
rate -> The repetition rate at which the variable is to
be accessed:
a value of 0 == > a one-time access
a value > 0 == > read the value every rate seconds
starting_time -> The time of day the start command was processed.
usage -> Direction of access on the variable.

pragma page;
procedure Deactivate (Vid: In Variable_Identifier;
Usage: In Io_Usage);

--/ Description:
--/ Informs the dialogue_manager that the variable is no longer
--/ of interest and data collection is not needed.
--/
--/ Parameter Description:
--/ vid -> The identifier of the variable to deactivate
--/ (obtained from get_identifier above).
--/ usage -> Direction of access on the variable being deactivated
--/ (since it is possible for a variable to activate in
--/ both directions).
--/------------------------------------------------------------------------

pragma page;
-- Exceptions rasied and propagated
--
-- Issued when a request is made for a variable that is not available
-- to the user through the RTM.
  Variable_Not_Found: exception;
--
-- Issued when a request is made to deposit the value of a variable, but the
-- value is not appropriate for the type of the variable.
  Illegal_Value: exception;

private

  type Active_List_Representation (Length_Of_Value: Positive);

  type Variable_Identifier is access Active_List_Representation;

end Dialogue_Manager;
pragma page;
Module Name: Dialogue_Manager
Module Type: Package Body

Module Description:
The dialogue_manager is responsible for the interface to the application, and as such, it needs to know a great deal more about the Ada variables than the user does. To do this work, the dialogue_manager is divided into three parts:

I. The dialogue_manager, which is the interface to all the services performed:
   - getting identifiers
   - activating variables
   - retrieving values

II. Collect_data package, that knows how to collect the data from the core and how to convert data into strings which represent a displayable value.
    This is a highly volatile package, which must have a conversion routine available for every possible Ada type which will be monitored.

References:
Design Documents:
- Real-Time Monitor Requirements
- Real-Time Monitor Design

User's Manual:
- RTM User's Manual

Testing and Validation:
- none

Notes:
- none

Modification History:
30Apr87 rvv created

Distribution and Copyright Notice:
TBD

Disclaimer:
This work was sponsored by the Department of Defense. The views and conclusions contained in this document are solely those of the author(s) and should not be interpreted as representing official policies, either expressed or implied, of Carnegie Mellon University, the U.S. Air Force.
the Department of Defense, or the U.S. Government.
with Address_Generator;
  -- Use the type "address_representation".

with Variable_Database;
  -- Use the type "the_variable".
  -- Use the services "find" and "initialize_database".

with Types_Manager;
  -- Use the services "convert_value_to_string", "convert_string_to_value", 
  -- and "get_type_information".

with Lists:
  -- This is a generic linked list package which is instantiated and used to 
  -- manipulate list objects.

with Sysgen;
  -- Need access to the sysgen parameters that control the interface to the 
  -- Rtm_Core.

package Dialogue_Manager is
  -- This package encapsulates the task runs asynchronously from the rest 
  -- of the monitor, collecting data from the application, and depositing 
  -- data into the application. The main reason for the package around the 
  -- task is to compensate for a VAX/VMS problem with elaborating subroutines 
  -- local to the body of a task at run time.

package Collect_Data is
  procedure From_Application;
  -- tbd task from application is
  -- tbd entry initiate;
  -- tbd entry get_next_set;
  -- tbd end from application;
end Collect_Data;

  --

  type Value_Representation is array (Positive range <>) of 
    Sysgen.Smallest_Unit;

  -- While the variable_database contains information about all the 
  -- variables in the system, the RTM is only concerned with a subset of these 
  -- variables at any one time. To keep track of the variables currently 
  -- relevant to the RTM, we introduced the concept of active variables. 
  -- The discussion below describes the implementation of active variables.

  -- Active_list_representation defines the data needed to periodically 
  -- access a single variable in the application:
  -- database_identifier -> Pointer to the variable's data in the 
    -- variable database.
  -- object_address -> Address of the variable/object at the time of 
    -- activation.
  -- update_rate -> The rate at which the variable is to be read.
  -- next_scheduled_reading -> The time of the next reading of the
variable out of application memory.

value -> The current value of the variable, as read from
       application memory or set by the user.
time_tag -> Time associated with the current value.

type Active_List_Representation (Length_Of_Value: Positive) is record
  Database_Identifier: Variable_Database.The_Variable;
  Object_Address: Address_Generator.Address_Representation;
  Update_Rate: Duration;
  Next_Scheduled_Reading: Time;
  Value: Value_Representation (1..Length_Of_Value) :=
       (1..Length_Of_Value => 0);
  time_tag: time;
end record;

Since we are almost always going to be working with several variables
at a time, we organize all the active variables into a linked list.
To do this, we must define an equality operator for list
items and instantiate a generic list package with the item representation
and the equality definition (defined further below). Now, we have a
linked list of pointers to the active variable representations.

function Equality (Left: Variable_Identifier;
       Right: Variable_Identifier) return Boolean;

package Active_Lists is new Lists (Itemtype => Variable_Identifier,
       Equal => Equality);

Once we have the list manipulator in place, we define two lists of
active variables:
active_read_list -> Holds all the variables currently
       being read by the RTM.
active_write_list -> Holds all the variables currently
       being written by the RTM.
Active_Read_List: Active_Lists.List := Active_Lists.Create;
Active_Write_List: Active_Lists.List := Active_Lists.Create;

package Collect_Data is separate;
pragma page;
function Equality (Left: Variable_Identifer;  
          Right: Variable_Identifer) return Boolean is

--/ Description:  
--/ This function defines the meaning of equality on the items  
--/ of type active_list_representation. We needed to define this  
--/ since normal equality would have compared two access values for  
--/ numeric equality (which would not have worked). The meaning  
--/ of equality of two entries in the active list is:  
--/ compare the value of the pointers in the active list,  
--/ if they match ==> the two items are pointing at the  
--/ same variable and are therefore  
--/ equal.  
--/
--/ Parameter Description:  
--/ left  ->  pointer to an active_list_representation record  
--/ right  ->  pointer to an active_list_representation record  
--/ return  ->  boolean,  
--/ true  ->  if the pointer component of both records point at  
--/ the same item  
--/ false  ->  if they point at different items  
--/ 
--/ Notes:  
--/ none  
--/----------------------------------------------------------------------------------------------------------

begin
    RETURN Left = Right;
end;

pragma page;
function Get_Value (Vid: In Variable_Identifier;
Number_Of_Characters: In Integer := 80)
    return Value_String is

-- Description:
-- Converts the current value of a variable (denoted as a
-- variable identifier) into a character string containing
-- the requested number of characters. This function returns
-- a string appropriate to the type of the variable.

-- Parameter Description:
-- vid -> The identifier of the variable whose value is needed
-- (obtained from get_identifier above).
-- number_of_characters -> The number of characters needed in the
-- value string.
-- return -> A string containing a displayable value. For a
-- composite structure, the individual components
-- are delimited by the separator selected.

-- Notes:
-- none

Value: Value_String;
begin
    Collect_Data.From_Application;
    Types_Manager.Convert_Value_To_String(Vid.Database_Identifier.Data_Type,
        Vid.Value(1)'Address,
        Number_Of_Characters,
        Value);

    RETURN Value;
end Get_Value;

pragma page;
function Get_Time_Of_Value (Vid: in Variable_Identifier) return Value_String is

--/ Description:
--- Convert the time associated with a variable value (i.e., the
--- collection time) into a displayable string.

--/ Parameter Description:
--- vid -> The identifier of the variable whose time is needed
--- (obtained from get_identifier above).
--- return -> The collection time of the current value as a
displayable string.

--/ Notes:
--- none

begin
   RETURN "";
end Get_Time_Of_Value;

pragma page;
procedure Set_Value (Vid: in Variable_Identifier;
   Value: in Value_String) is

   --/ Description:
   --/ Sets the value of the selected variable, normally in preparation
   --/ for a set (write) operation. The types_manager converts
   --/ the string entered by the user into internal representation.
   --/
   --/ Parameter Description:
   --/ vid -> The identifier of the variable whose value is being set
   --/ (obtained from get_identifier above).
   --/ value -> A string containing the value to deposit into
   --/ the variable internal representation. For composite
   --/ structure, the separator selected.
   --/
   --/ Notes:
   --/ none
   --/
   New_Value: Value_Representation(1..Vid.Length_Of_Value);
   begin
      Types_Manager.Convert_String_To_Value(Vid.Database_Identifier.Data_Type,
         New_Value'Address,
         Value);
      Vid.Value := New_Value;
   exception
      when Types_Manager.Illegal_Value =>
         RAISE Illegal_Value;
      when others =>
         RAISE;
   end Set_Value;

pragma page;
function Activate (Name: In String; Rate: In Duration; Starting_Time: Time; Usage: In lo_Usage) return Variable_Identifier is

--/ Description:
--/ This entry activates variables for data collection or modification.
--/ It builds the active variable representations used by
--/ the collect_data package in gathering information from the
--/ application. Basically, all the data needed to access
--/ a variable are passed in as parameters. Activate builds a
--/ record from the data and adds it to the current list of active
--/ variables.

--/ Parameter Description:
--/ name -> The name of the variable to activate.
--/ rate -> The repetition rate at which the variable is
--/ to be accessed:
--/  a value of 0 => a one-time access
--/  a value > 0 => read the value every rate seconds
--/ starting_time -> The time of day the start command was processed.
--/ usage -> Direction of access on the variable.
--/ return  -> The identifier of the variable to be activated.

--/ Notes:
--/ Activate operates on both the read and write lists.
--/ Currently, the RTM is not sophisticated enough to understand
--/ the complexities of offsets as encountered in arrays, for example.
--/ Thus, the offset is always returned as a 0. This means that each
--/ element of an array must explicitly have an entry in the variable
--/ database. This is solely a restriction on the prototype RTM, not
--/ a restriction on the concepts involved.

Data_Length: Integer;
Vid: Variable_Identifier;
Access_Flag: Boolean;
Active_Variable: Variable_Database.The_Variable;
Address: Address_Generator.Address_Representation;
begin
--/ Determine the system address for the variable/object.

Active_Variable := Variable_Database.Find(Name);
Types_Manager.Get_Type_Information (Active_Variable.Data_Type,
  Data_Length,
  Access_Flag);
Address := Address_Generator.Compute_Address(Variable_Name => Name);
--/ Build the activation record for the variable.

Vid := new Active_List_Representation(Data_Length);
Vid.Object_Address := Address;
Vid.Database_Identifier := Active_Variable;
Vid.Update_Rate := Rate;

145 October 1987
Vid.Next_Scheduled_Reading := Starting_Time+Rate.

case Usage is
  when Read =>
    Active_Lists.Attach (Active_Read_List.Vid);
  when Write =>
    Active_Lists.Attach (Active_Write_List.Vid);
end case;

RETURN Vid;

exception
  when Variable_Database.Variable_Not_Found =>
    RAISE Variable_Not_Found;
  when others =>
    RAISE;
end Activate;

pragma page;
procedure Deactivate (Vid: in Variable_Identifier,  
   Usage: in lo_Usage) is  

--/ Description:  
--/ Takes the identifier (and list) of an active variable and  
--/ deletes it from the active variable list. This operation  
--/ destroys the entry in the list, but doesn't affect the data  
--/ about the variable anywhere else.  
--/  
--/ Parameter Description:  
--/ vid -> The identifier of the variable to deactivate  
--/ (obtained from get_identifier above).  
--/ usage -> direction of access on the variable being deactivated  
--/ (since it is possible for a variable to activate in  
--/ both directions).  
--/  
--/ Notes:  
--/ none  
--/  
begin  
case Usage is  
   when Read =>  
      Active_Lists.DeleteItem(Active_Read_List, Vid);  
   when Write =>  
      Active_Lists.DeleteItem(Active_Write_List, Vid);  
end case;  
end Deactivate;  

--/ Dialogue_manager package body  
--/  
--/ The body has one startup operation:  
--/ Initialize the variable database  
--/  
begin  
   Variable_Database.Initialize_Database;  
--/ tbd collect data from application initiate;  
end Dialogue_Manager;  
pragma page;
---

**Module Name:**
- Collect_Data

**Module Type:**
- Package Body

**Module Description:**
- This package implements the functions which interface to the
  RTM_core and do the actual data collection.

**References:**
- **Design Documents:**
  - RTM Design Description
  - RTM Design

- **User's Manual:**
  - RTM User's Manual

- **Testing and Validation:**
  - none

- **Notes:**
  - none

**Modification History:**
- 22May87  rvs created

**Distribution and Copyright Notice:**
- TBD

**Disclaimer:**
- "This work was sponsored by the Department of Defense.
- The views and conclusions contained in this document are
- solely those of the author(s) and should not be interpreted as
- representing official policies, either expressed or implied,
- of Carnegie Mellon University, the U.S. Air Force,
- the Department of Defense, or the U.S. Government."

---

Pragma page;
with Rtm_Core;
-- Use the type "rtm_core_command_representation".
--
with Calendar;
-- Use the type "time".
--
separate (Dialogue_Manager)

package Collect_Data is
--
-- This structure allows us map the data returned by the RTM_core
-- back into the variable database. It consists of:
-- vid -> The pointer into the variable database where the
-- variable's value is kept.
--
  Result_Map: array (2..Sysgen.Core_Buffer_Size) of Variable_Identifier;
--
-- These structures map from the data_types of variables into the Rtm_Core
-- commands needed to access the type.
--
  type Core_Operation_Representation is (Deposit, Extract);
  type Command_Map_Representation is array
    (Core_Operation_Representation) of
      Rtm_Core.Rtm_Core_Command_Representation;
  Command_Map: Command_Map_Representation :=
    (Deposit => Rtm_Core.Dposit,
     Extract => Rtm_Core.Extract);
--
--
  Next_Update_Time: Calendar.Time;
--
-- Internal procedures
--
procedure Build_Rtm_Core_Commands ( List: in out Active_Lists.List;
  Command: in Core_Operation_Representation;
  Command_Position: in out Rtm_Core.Buffer_Range;
  Data_Position: in out Rtm_Core.Buffer_Range) is separate;

procedure Retrieve_Rtm_Core_Results (Ending_Position: in Integer) is separate;
--
-- Package procedures
--
procedure From_Application is separate;

end Collect_Data;
pragma page;
with Rtm_Core;
-- Use the service "process_buffer".
...

separate (Dialogue_Manager.Collect_Data)
procedure From_Application is
-- tbd Task body from application is
-- ..............................................................
-- Description:
-- 1. Building a list with all the deposit (Set) commands to be done.
-- 2. Adding to that list all the extract (Read or Start) commands
-- to be done.
-- 3. Calling the RTM_Core to process the commands and waiting
-- for it to complete.
-- 4. Retrieving the results of the commands and storing
-- them in the variable database.
--
-- Parameter Description:
-- none
--
-- Notes:
-- Next_Command_Position: Rtm_Core.Buffer_Range := 1;
-- Next_Data_Position: Rtm_Core.Buffer_Range := 1;
begin
-- tbd Accept initiate;
-- tbd next_update_time := calendar.clock + sysgen.minimum_delay;
-- tbd loop
-- tbd delay next_update_time - calendar.clock;
-- tbd Accept get_next_set do
-- If Calendar.Clock < Next_Update_Time then
-- RETURN;
-- end If;
Build_Rtm_Core_Commands
(List => Active_Write_List,
 Command => Deposit,
 Command_Position => Next_Command_Position,
 Data_Position => Next_Data_Position);
Build_Rtm_Core_Commands
(List => Active_Read_List,
 Command => Extract,
 Command_Position => Next_Command_Position,
 Data_Position => Next_Data_Position);
Rtm_Core.Process_Buffer;
Retrieve_Rtm_Core_Results (Ending_Position => Next_Command_Position);
-- tbd end get_next_set;
-- tbd End loop;
end From_Application;
pragma page;
procedure Build_Rtm_Core_Commands ( 
List in out Active_Lists.List;
Command in Rtm_Core_Operation_Representation;
Command_Position in out Rtm/Core.Buffer_Range;
Data_Position in out Rtm/Core.Buffer_Range) is

--/ Description:
--/ This module takes a list of active variables and a command
--/ to be associated with those variables and builds a command
--/ buffer for the Rtm_Core to process. It does this by:
--/ 1. Looping through all the variables in the active list.
--/ 2. Checking the next operation time of each variable,
--/ and if the time has come:
--/   a. for each piece of the variables value,
--/      - format the command triplet:
--/        (command, address, value/status)
--/      - format the map entry to retrieve the data with
--/Parameter Description:
--/  list  -> The active variable list from which the
--/  command  -> The Rtm_Core command to be formatted.
--/  command_position  -> The beginning point in the
--/                   command buffer where the commands
--/                    are to be placed.
--/  data_position  -> The beginning point in the
--/                    data buffer where the data are (to be)
--/                    stored.

--/ Notes:
--/  command_position & data_position must be initialized and
--/  passed into this procedure to insure proper functioning.
--/  Once passed in, these parameters are modified and returned so
--/  that successive calls to this procedure can incrementally build
--/  the command buffer.

begin
-- We operate on this list in a slightly different way. We get
-- the size of the list before we begin operating on the list, and we
-- build a list iterator to get elements from the list. Then, we loop
-- the list checking time of each entry. If the scheduled time has arrived,
-- we format the command and move the entry to the end of the list, the
-- reason being that we can only fit a finite number of commands in the
-- buffer at any one time; this movement insures that commands missed on
-- one pass will be picked up on a subsequent pass.

List_Size := Active_Lists.Length(List);
List_Position := Active_Lists.MakeListIter(List);
for Count in 1..List_Size loop
    Active_Lists.Next(List_Position, The_Next_Variable);
    Data_Count = The_Next_Variable.Length_Of_Value.

    if (Calendar.Clock >= The_Next_Variable.Next_Scheduled_Reading) and
        (Data_Position + Data_Count <= Sysgen.Core_Buffer_Size) then
        -- Build the next command for the core.
        Command_Position := Command_Position + 1;
        Rtm_Core.Command_Buffer(Command_Position).Command :=
            Command_Map(Command);
        Rtm_Core.Command_Buffer(Command_Position).Data_Address :=
            The_Next_Variable.Object_Address;
        Rtm_Core.Command_Buffer(Command_Position).Data_Count :=
            Data_Count;
        Rtm_Core.Command_Buffer(Command_Position).Data_Location :=
            Data_Position;
        -- Build the map to extract the results later.
        Result_Map(Command_Position) := The_Next_Variable;
        -- Fill in the data to transfer to the application (for deposit commands).
        case Command is
            when Deposit =>
                for Next in 1..Data_Count loop
                    Rtm_Core.Data_Buffer(Data_Position) :=
                        The_Next_Variable.Value(Next);
                    Data_Position := Data_Position + 1;
                end loop;
            when Extract =>
                Data_Position := Data_Position + Data_Count;
        end case;
        -- Mark the command buffer as ready for processing.
        Rtm_Core.Command_Buffer(1).Command := Rtm_Core.Buffer_Available;
        -- Move the element just processed to the end of the list.
        Active_Lists.Deleteitem (L => List,
            Element => The_Next_Variable);
        The_Next_Variable.Next_Scheduled_Reading :=
            The_Next_Variable.Next_Scheduled_Reading +
            The_Next_Variable.Update_Rate;
        Active_Lists.Attach (List,The_Next_Variable);
    end if;
    -- Finally, keep a running tab on the next scheduled update time for all
    -- the variables. This will prevent us from doing any unneeded
    -- list traversals.
If Next_Update_Time > The.Next_Variable.Next_Scheduled_Reading then
    Next_Update_Time := The.Next_Variable.Next_Scheduled_Reading;
end if;
end loop;
end Build_Rtm_Core_Commands;
pragma page;
procedure Retrieve_Rtm_Core_Results (Ending_Position: in Integer) is
-- Description:
-- This procedure maps the data collected by the Rtm_Core
-- back into the active variable list.
--
-- Parameter Description:
-- ending_position -> Marks the position of the last command
-- stored in the command buffer.
--
-- Notes:
-- none
begin
Next_Data_Location := Rtm_Core.Buffer_Range := 1;
begin
-- Loop through all the commands in the core buffer, mapping
-- the data back into the active variable list.
--
for Position in 2..Ending_Position loop
  for Next in 1..Rtm_Core.Command_Buffer(Position).Data_Count loop
    Result_Map(Position).Value(Next) :=
      Rtm_Core.Data_Buffer(Next_Data_Location);
    Next_Data_Location := Next_Data_Location + 1;
  end loop;
end loop;
end Retrieve_Rtm_Core_Results;
pragma page;
Module Name: Sysgen

Module Type: Package Specification

Module Purpose:
Define the system-wide constants needed when rehosting and tuning the RTM.

Module Description:
This package defines the system-dependent constants needed by the RTM. All the constants are completely described below.

References:
Design Documents:
- RTM Design Description

User's Manual:
none

Testing and Validation:
none

Notes:
none

Modification History:
02Apr87 rlv Created

Distribution and Copyright Notice:
TBD

Disclaimer:
"This work was sponsored by the Department of Defense. The views and conclusions contained in this document are solely those of the author(s) and should not be interpreted as representing official policies, either expressed or implied, of Carnegie Mellon University, the U.S. Air Force, the Department of Defense, or the U.S. Government."

pragma page;
package Sysgen is
  -- Defines the minimum amount of time between successive updates of the
  -- screen.
  --
  -- Minimum_Delay: constant Duration := 0.1;
  -- Defines the smallest addressable unit on the RTM_core CPU.
  --
  -- subtype Smallest_Unit is Integer;
  -- Defines the maximum number of commands which the RTM_core can
  -- process in one time-slice.
  --
  -- Core_Buffer_Size: constant := 1000;
  -- Defines the number of processors in the RTM/application configuration.
  --
  -- Processor_Count: constant := 1;
  -- Defines the default disk where the RTM can access stored information.
  --
  -- Default_Rtm_Device: constant String := "ps[rtm.prototype.rtm]";
end Sysgen;
pragma page;
Module Name: Rtm_Core

Module Type: Package Specification

Module Purpose:
Performs read (extract) and write (deposit) operations on system storage units, which are the smallest addressable units in the system.

Module Description:
This package is an abstraction for the actual application software underlying the RTM. This allows the RTM to know how to talk with the Rtm_Core, but relieves it of the need to know anything about the specific application.

References:
Design Documents:
- Real-Time Monitor Requirements
- Real-Time Monitor Design

User's Manual:
- RTM User's Manual

Testing and Validation:
none

Notes:
There are two buffers that form the interface between the RTM and the application: the command_buffer, shown below, which holds all the command and address information needed to perform the requested operations and the data_buffer, which simply holds the data to deposit or the data extracted. These two buffers are connected by the <data_location> field shown below.

Command Buffer

<table>
<thead>
<tr>
<th>&lt;buffer available&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;deposit marker&gt;</td>
</tr>
<tr>
<td>&lt;address&gt; (base address, offset, flag)</td>
</tr>
<tr>
<td>&lt;data count&gt;</td>
</tr>
<tr>
<td>&lt;data location&gt;</td>
</tr>
<tr>
<td>&lt;extract marker&gt;</td>
</tr>
</tbody>
</table>
This function is the partition point of the monitor in a multi-processor implementation. The core function described here is all of the monitor that MUST be part of the application timing and control.

The size of the command and data buffers is a sysgen parameter which can be tuned to meet the performance needs of the system.

The details on how the partition between one and two processors would take place are discussed in the design description document for the RTM.

Modification History:
02Apr67  rlv Created

Distribution and Copyright Notice:
TBD

Disclaimer:
"This work was sponsored by the Department of Defense. The views and conclusions contained in this document are solely those of the author(s) and should not be interpreted as representing official policies, either expressed or implied, of Carnegie Mellon University, the U.S. Air Force, the Department of Defense, or the U.S. Government."

Pragma page;
with Address_Generator;
-- Uses the type "address_representation".
with Sysgen;
-- Uses "core_buffer_size", which is the system parameter that controls
-- the maximum size of the interface buffer to the core and thereby controls
-- the maximum amount of processing the core will need.
-- Uses "smallest_unit", which defines the smallest addressable unit that
-- can be read or written.

package Rtm_Core is
-- Define the legal commands which will be recognized;
-- these are defined as an integer subtype because in a two-CPU
-- configuration, there is knowledge on either CPU about the
-- representation of data on the other CPU. Therefore, integers are
-- the safest means of communicating commands across a bus.

subtype Rtm_Core_Command_Representation is Positive range 1..10;
Address_Error: constant Rtm_Core_Command_Representation := 10;
Buffer_Available: constant Rtm_Core_Command_Representation := 9;
Results_Available: constant Rtm_Core_Command_Representation := 8;
Masked_Deposit: constant Rtm_Core_Command_Representation := 7;
Masked_Extract: constant Rtm_Core_Command_Representation := 6;
Deposit: constant Rtm_Core_Command_Representation := 3;
Extract: constant Rtm_Core_Command_Representation := 2;
End_Of_Buffer: constant Rtm_Core_Command_Representation := 1;

-- This object makes two arrays visible to the external world,
-- the reason being that in a single processor system, the object commanding
-- the Rtm_Core can do so by simply filling in the buffers before invoking it.
-- In a multi-processor environment, the buffers are visible to
-- the bus I/O handler. The command_buffer is filled with as many
-- deposit/extract commands as will fit. The only requirements on the
-- buffer are that it start with a buffer_available command and that the
-- last command be followed by an end_of_buffer command. Also,
-- no usable data in the data_buffer are sent back to the RTM,
-- until the buffer_available command has been overwritten by a
-- results_available command in the command_buffer.

subtype Buffer_Range is Integer range 1..Sysgen.Core_Buffer_Size;
type Buffer_Entry_Representation is record
  Command: Rtm_Core_Command_Representation := End_Of_Buffer;
  Data_Address: Address_Generator.Address_Representation;
  Data_Count: Buffer_Range;
  Data_Location: Buffer_Range;
end record;

Command_Buffer: array (1..Buffer_Range'Last) of
  Buffer_Entry_Representation := (others =>
    (End_Of_Buffer, Address_Generator.Null_Address, 1, 1));
Data_Buffer: array (1..Buffer_Range'Last) of
  Sysgen.Smallest_Unit := (others => 0);
procedure Process_Buffer;

-- Description:
-- Instructs the core to check its communications buffer for commands.
-- If there are commands available, then process the buffer. If not,
-- no processing action is taken.

-- Parameter Description:
-- none

end Rtm_Core;
pragma page;
Module Name: RTM_Core

Module Type: Package Body

Module Description:
This module processes the commands formatted by the RTM, by converting integers from the command buffer into pointers and then using the pointers to access the data.

References:
Design Documents:
- Real-Time Monitor Requirements
- Real-Time Monitor Design

User's Manual:
- RTM User's Manual

Testing and Validation:
none

Notes:
none

Modification History:
16Apr87 rvs created

Distribution and Copyright Notice:
TBD

Disclaimer:
"This work was sponsored by the Department of Defense. The views and conclusions contained in this document are solely those of the author(s) and should not be interpreted as representing official policies, either expressed or implied, of Carnegie Mellon University, the U.S. Air Force, the Department of Defense, or the U.S. Government."

pragma page;
with System;
-- Need access to the type "address".
--
with Unchecked_Conversion;
-- Need "unchecked_conversion" to convert the integer in the
-- command buffer into a pointer to a value of type "sysgen.smallest_unit".
--
package Rtm_Core is
--
-- Set up the system work needed to access data on the core CPU.
-- First, we create a type that points to a value of the smallest
-- addressable unit on the CPU. Then, we instantiate unchecked_conversion
-- to allow us to transform the integer form of the address in the command
-- buffer into an address that Ada will understand.
--
  type Value_Pointer is access Sysgen.Smallest_Unit;
  function Get_Address is new Unchecked_Conversion
    (Source => Integer,
     Target => Value_Pointer);
  function Get_Actual_Address is new Unchecked_Conversion
    (Source => Sysgen.Smallest_Unit,
     Target => Value_Pointer);
pragma page;
---

--- internal procedures
---

procedure Compute_Address (Data_Address out Integer,
Command_Number in Integer) is

---/ Description:
---/ This module is responsible for decoding the address parameter
---/ of the command passed in. This is a two-step operation:
---/ 1. For indirect addresses, the actual base address
---/    must be read.
---/ 2. The offset must be added to the base address.
---/

---/ Parameter Description:
---/ data_address  -> The computed address of the desired data.
---/ command_number -> Command being processed in the command_buffer.
---/

---/ Notes:
---/ none
---/................................................................

Address: Address_Generator.Address_Representation
  renames Command_Buffer(Command_Number).Data_Address;
Value_Address: Value_Pointer;
Actual_Base_Address: Integer;
begin
  if Address.Indirection then
    Value_Address := Get_Address(Address.Base_Address);
    Actual_Base_Address := Integer(Value_Address.all);
    Data_Address := Actual_Base_Address + Address.Address_Offset;
  else
    Data_Address := Address.Base_Address + Address.Address_Offset;
  end if;
end Compute_Address;
pragma page;

---
procedure Deposit_Data (Data_Address: In Integer;
    Command_Number: In Buffer_Range) is

.. ........................................
-- /Description:
-- / Moves the data from the data_buffer passed by the RTM into
-- / application memory.
-- /
-- / Parameter Description:
-- / data_address -> The computed address of the desired data
-- /     In the case of a multiple unit read, this
-- /     is the address of the first unit in the block.
-- / command_number -> Command being processed in the command_buffer
-- /
-- / Notes:
-- / none
-- ........................................

Next_Address: Integer := Data_Address;
The_Value: Value_Pointer := Get_Address(Next_Address);
Data_Offset: Buffer_Range renames Command_Buffer(Command_Number).Data_Location;
begin
    for Next In 0..Command_Buffer(Command_Number).Data_Count-1 loop
        The_Value.all := Data_Buffer(Next + Data_Offset);
        Next_Address := Next_Address + 1;
        The_Value := Get_Address(Next_Address);
    end loop;
end Deposit_Data;

pragma page:
procedure Extract_Data (Data_Address: In Integer;
Command_Number: in Buffer_Range) is
--/ Description:
--/ Moves the data from application memory into data_buffer passed
--/ back to the RTM.
--/
--/ Parameter Description:
--/ data_address  -> The computed address of the desired data.
--/ In the case of a multiple unit read, this
--/ is the address of the first unit in the block.
--/ command_number  -> Command being processed in the command_buffer.
--/
--/ Notes:
--/ none
--/
Next_Address: Integer := Data_Address;
The_Value: Value_Pointer := Get_Address(Next_Address);
Data_Offset: Buffer_Range renames Command_Buffer(Command_Number).Data_Location;
begin
for Next in 0..Command_Buffer(Command_Number).Data_Count-1 loop
  Data_Buffer(Next + Data_Offset) := The_Value.all ;
  Next_Address := Next_Address + 1;
  The_Value := Get_Address(Next_Address);
end loop ;
end Extract_Data;

pragma page;
procedure Get_Buffer is

--/ Description:
--/ This procedure is available for a two-CPU implementation
--/ of the RTM. It is responsible for knowing how to receive
--/ data over a communications bus and doing any conversions needed
--/ to get the data into a usable format.

--/ Parameter Description:
--/ none

--/ Notes:
--/ none

begin
    case Sysgen.Processor_Count is
        when 1 =>
            null;
        when 2 =>
            null;
        when others =>
            null;
    end case;
end Get_Buffer;

pragma page;
procedure Send_Buffer is
--

Description:
--
This procedure is available for a two-CPU implementation
--
of the RTM. It is responsible for knowing how to send
--
data over a communications bus and doing any conversions needed
--
to get the data into a usable format before transmission.
--

Parameter Description:
--
one
--

Notes:
--
one
--

begin
  case Sysgen.Processor_Counter is
  when 1 =>
    null;
  when 2 =>
    -- Here is where the bus i/o goes
    null;
  when others =>
    null;
  end case;
end Send_Buffer;

pragma page;
procedure Process_Buffer Is

-- Description:
-- This module reads the command buffer from the interface
-- (which doesn't exist on a one-CPU implementation) and loops
-- through the commands performing the operations. Currently there
-- are four operations defined:
-- masked_deposit -> Requests that the core move a value
-- from the command buffer into application
-- memory, but mask the target so that unaffected
-- bits will be preserved (for rep clauses);
-- masked_extract -> Requests that the core move a value
-- from application memory into the command
-- buffer, but mask the target so that unaffected
-- bits will be preserved (for rep clauses);
-- deposit -> Requests that the core move a value
-- from the command buffer into application memory.
-- extract -> Requests that the core move a value
-- from application memory into the command buffer.
-- When all the commands in the buffer have been processed, the
-- buffer_available command of the RTM is overwritten by the
-- results_available command of the core.
--
-- Parameter Description:
-- none
--
-- Notes:
-- If the core is unable to use an address contained in the
-- command buffer, the command associated with that address
-- is overwritten by an address_error command.
--
-- The masked_extract and masked_deposit operations are not currently
-- implemented.
--
Value_Address: Integer;

begin
-- Load the buffer, and check to see if any new commands are
-- available for processing. If not, we simply cut out without
-- further ado.
--
Get_Buffer;
if (Command_Buffer(1).Command /= Buffer_Available) then
  return;
else

-- When there are commands to be processed, we loop through the command
-- buffer, processing commands until an end_of_buffer is found, or
-- we reach the end of the buffer
--
for Command In 2..Sysgen.Core_Buffer_Size loop
  case Command_Buffer(Command).Command Is
    when Deposit =>
      Compute_Address (Value_Address, Command);
  end case;
end loop;

Deposit_Data (Value_Address, Command);
when Extract =>
  Compute_Address (Value_Address, Command);
  Extract_Data (Value_Address, Command);
when Masked_Extract =>
  null;
when End_Of_Buffer =>
  EXIT;
when others =>
  null;
end case;
end loop;

-- Mark the results as being available and return them to the RTM
--
  Command_Buffer(1).Command := Results_Available;
  Send_Buffer;
end Process_Buffer;

end Rtm_Core;
Index

Activate, function 134, 145  
Active_Lists, type 136, 140, package 140  
Address_Generator, package 126  
Address_Generator, package body 127  
Address_Generator, with 139, 159  
Address_Representation, type 140, 141  
Address_To_enum_Pointer, function 89  
Address_To_Float_Pointer, function 97  
Address_To_Integer_Pointer, function 81, 85, 87  
Address_To_Rtm_Enum_Pointer, function 97  
App, procedure 1  
Binarytrees, with 110  
Buffer_Entry_Representation, type 159  
Buffer_Range, subtype 159  
Build_Rtm_Core_Commands, procedure 149, 151  
Calendar, with 41, 65, 67, 130, 149  
CaseInsensitive_String_Comparison, with 49, 104, 110  
Check_Page, procedure 36, 43, 44  
Cisc, package 49, 104, 110  
Clear_Command_Line, procedure 28, 30, 34  
Clear_Rtm_Field, procedure 15, 23  
Cli, package 45, 47, 49, 52, 57, 65, 67  
Closeout_Rtm, procedure 6, 10  
Collect_Data, package 139  
Collect_Data, package body 140, 149  
Command_Map_Representation, type 149  
Compute_Address, function 126, 127, 128  
Compute_Address, procedure 164  
Conversions, package 71  
Conversions, package body 79  
Conversions, with 96  
Convert_Enumerations, generic package 77  
Convert_Enumerations, package body 79, 89  
Convert_Floats, generic package 75  
Convert_Floats, package body 79, 85  
Convert_Integers, generic package 73  
Convert_Integers, package body 79, 81  
Convert_String_To_Value, procedure 93, 101, 103  
Convert_Type_To_String, procedure 93, 101, 102  
Core_Operation_Representation, type 149  
Db, package 111  
Deactivate, procedure 135, 147  
Default_Float_Conversion, function 98  
Default_Integer_Conversion, function 99  
Define_Rtm_C, package 27  
Define_Rtm_Cli, package 30  
Define_Rtm_Cli, with 6, 41, 65, 67  
Deposit_Data, procedure 165  
Dialogue_Manager, package 130  
Dialogue_Manager, package body 139  
Dialogue_Manager, with 41, 44, 56, 65, 67  
Duration_10, package 41  
Enum_Pointer, type 89  
Enum_Pointer_To_Address, function 89  
Extract_Data, procedure 166  
Field_Lists, package 42  
Find, function 93, 101, 104, 108, 112  
Float_Pointer, type 97  
Form_Executor, with 18, 41, 44, 56  
Form_Manager, with 18, 41, 44, 54, 56  
Free, procedure 113  
Form_Application, procedure 139, 149, 150  
Function  
Activate 134, 145  
Address_To_Enum_Pointer 89  
Address_To_Float_Pointer 97  
Address_To_Integer_Pointer 81, 97  
Address_To_Rtm_Enum_Pointer 97  
Address_To_Real_Pointer 85  
Address_To_Float_Pointer 97  
Address_To_Integer_Pointer 81, 97  
Address_To_Real_Pointer 85  
Address_To_Rtm_Enum_Pointer 97  
Compute_Address 126, 127, 128  
Default_Float_Conversion 98  
Default_Integer_Conversion 99  
Enum_Pointer_To_Address 89  
Equality 140, 141  
Find 93, 101, 104, 108, 112  
Get_Actual_Address 163  
Get_Address 120, 163  
Get_Time_Of_Value 132, 143  
Get_Value 131, 142  
Integer_Pointer_To_Address 81  
Left_Justify 64, 69  
More 118, 124  
Ordering 111  
Real_Pointer_To_Address 85  
Rtm_Enum_Conversion 100  
Setup_Page 43, 56  
Generic package  
Convert_Enumerations 77  
Convert_Floats 75  
Convert_Integers 73  
Get_Actual_Address, function 163  
Get_Address, function 120, 163  
Get_Argument, procedure 28, 30, 33  
Get_Buffer, procedure 167  
Get_Fields, procedure 43, 54  
Get_Next, procedure 118, 122  
Get_Rtm_Field, procedure 14, 20  
Get_Time_Of_Value, function 132, 143  
Get_Type_Information, procedure 94, 101, 105  
Get_Value, function 131, 142  
Go, procedure 2  
Initialize_Database, procedure 108, 111, 113  
Initialize_Rtm_Form, procedure 14, 19  
Input_Mode, type 14  
Integer_Pointer, type 81, 97  
Integer_Pointer_To_Address, function 81  

171 October 1987
Interact, with 11
Internal_io, package 81, 85, 89
Io_Useage, type 130

Left_Justify, function 64, 69
Library_Interface, package 117
Library_Interface, package body 120
Library_Interface, with 108
Lists, with 41, 139

Make_Iterator, procedure 118, 121
Make_String, procedure 73, 75, 77, 82, 86, 90
Make_Value, procedure 73, 75, 77, 82, 86, 87, 91
More, function 118, 124

Ordering, function 111

Package
Active_Lists 140
Address_Generator 126
Cisc 49, 104, 110
Cl 45, 47, 49, 52, 57, 65, 67
Collect_Data 139
Conversions 71
Db 111
Define_Rtm_Cli 27
Dialogue_Manager 130
Duration_IO 41
Field_Lists 42
Internal_io 81, 85, 89
Library_Interface 117
Page_Processor 35
Parameter_Manager 61
Real_Time_Monitor 4
Rtm_Cli 27
Rtm_Core 159
RtmEnums 100
Rtm_Form 14
Rtm_Integers 99
Rtm_Reals 98
Sa 30
Si 30
Sp 33
Sysgen 156
Test_Sub 2
Types_Manager 93
Variable_Database 108

Package body
Address_Generator 127
Collect_Data 140, 149
Conversions 79
Convert_Enumerations 79, 89
Convert_Floats 79, 85
Convert_Integers 79, 81
Define_Rtm_Cli 30
Dialogue_Manager 139
Library_Interface 120
Page_Processor 41
Parameter_Manager 64
Real_Time_Monitor 6
Rtm_Core 163
Rtm_Form 18
Test_Sub 2
Types_Manager 96
Variable_Database 110
Page_Field_Representation, type 41
Page_Name_Representation, subtype 42
Page_Processor, package 35
Page_Processor, package body 41
Page_Processor, with 11
Page_Name_Representation, type 42
Parameter_Manager, package 61
Parameter_Manager, package body 64
Parameter_Manager, with 11
Procedure

Appl 1
Build_Rtm_Core_Commands 149, 151
Check_Page 36, 43, 44
Clear_Command_Line 28, 30, 34
Clear_Rtm_Field 15, 23
Closeout_Rtm 6, 10
Compute_Address 164
Convert_String_To_Value 93, 101, 103
Convert_Value_To_String 93, 101, 102
Deactivate 135, 147
Deposit_Data 165
Extract_Data 166
Free 113
From_Application 139, 149, 150
Get_Argument 28, 30, 33
Get_Buffer 167
Get_Fields 43, 54
Get_Next 118, 122
Get_Rtm_Field 14, 20
Get_Type_Information 94, 101, 105
Go 2
Initialize_Database 108, 111, 113
Initialize_Rtm_Form 14, 19
Make_Iterator 118, 121
Make_String 73, 75, 77, 82, 86, 90
Make_Value 73, 75, 77, 83, 87, 91
Process_Buffer 161, 169
Process_The_Command 6, 11
Put_Rtm_Field 14, 22
Read 61, 64, 65
Retrieve_Rtm Core_Results 149, 154
Rtm 4, 6, 7
Send_Buffer 168
Set 62, 64, 67
Set_Input_Mode 15, 24
Set_Value 133, 144
Setup_Rtm 6, 9
Start_Page 37, 43, 47
Stop_Page 38, 43, 49
Update_Pages 39, 43, 52
Process_Buffer, procedure 161, 169
Process_The_Command, procedure 6, 11
Put_Rtm_Field, procedure 14, 22

Read, procedure 61, 64, 65

October 1987
Form_Executor 18, 41, 44, 56
Form_Manager 18, 41, 44, 54, 56
Interact 11
Library_Interface 108
Lists 41, 139
Page_Processor 11
Parameter_Manager 11
Real_Time_Monitor 1
Rtm_Core 149, 150
Rtm_Form 6, 44, 47, 49, 56, 65, 67
Standard_Interface 7, 27, 30
String_Pkg 33
Sysdep 18
Sysgen 139, 159
System 71, 93, 120, 163
Terminal_Interface 6, 11, 49
Test_Stub 1, 96, 120
Text_lo 1, 41, 81, 85, 89
Types_Manager 113, 117, 127, 139
Unchecked_Conversion 81, 85, 89, 96, 120, 163
Unchecked_Deallocation 113
Variable_Database 44, 56, 126, 139
END
DATE
FILMED
5-88
PTIC